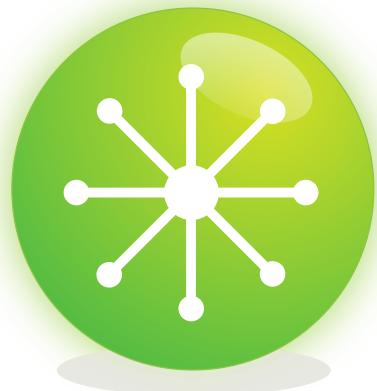


GTNA

A Framework for the Graph-Theoretic Network Analysis



Benjamin Schiller Dirk Bradler Immanuel Schweizer
Max Mühlhäuser Thorsten Strufe



GTNA

Graph-Theoretic Network Analyzer

Outline

-
1. Introduction
 2. Modules and Workflow
 3. Available Networks
 4. Available Metrics
 5. Extension
 6. Evaluation
 7. Summary and Outlook

1. Introduction

Motivation, Our Approach and Requirements



GTNA
Graph Theoretic Network Analyzer

Motivation



- Importance of complex networks rises
- Need for analysis and evaluation
- Simulation research tool of choice
- BUT: credibility of the results decreases

"... less than 15% of the published MobiHoc papers are repeatable."

Kurkowski et al. - MANET Simulation Studies: The Incredibles, 2006

omnipresent in our daily life

- Internet, P2P, WSN
- Mobile ad hoc networks

different simulation tools exist
BUT

- comparison hardly possible
- results differ significantly

survey simulation studies of MobiHoc
(= ACM international Symp. on Mobile
Ad Hoc Networking and Computing)

Main Objectives



- Many different networks and metrics

snapshots don't capture dynamics
BUT: allow for structural comparison

- Graph-theoretic analysis of complex networks

- Need for an analysis framework

many different implementations exist
=> need for a comprehensive, extendible tool
for the concise evaluation of graphs

- Plugin interface for metrics and network generators

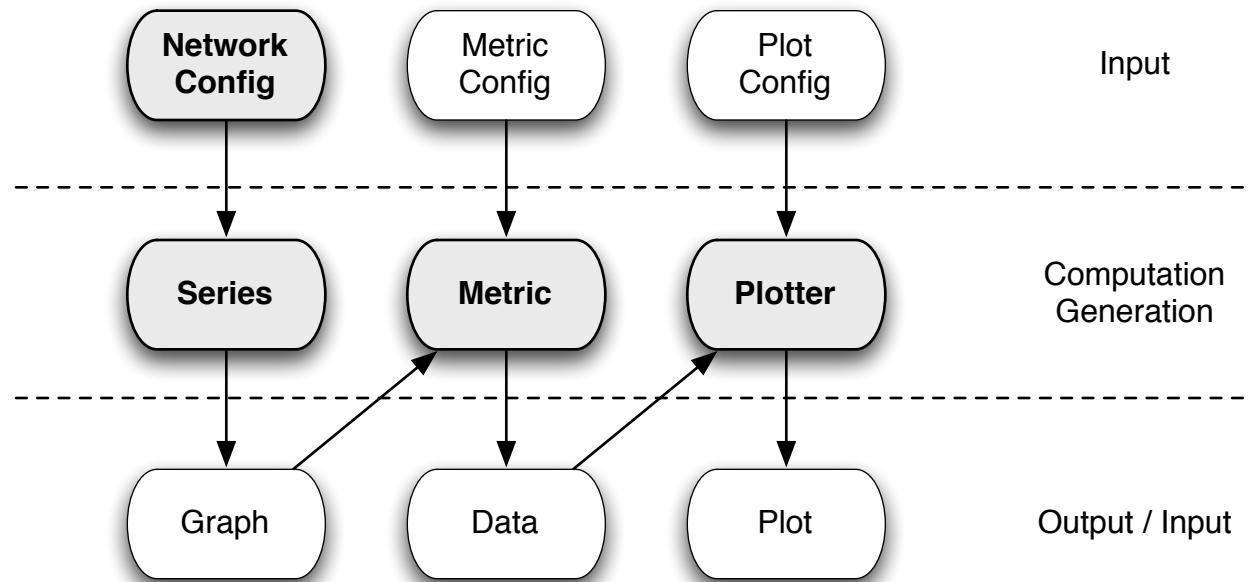
- plugin mechanism for
input data
topologies from other simulators
- extendable metrics analysis mechanism

Requirements

1. Extensability
2. Run on regular desktop computers
3. Evaluate networks with > 20.000 nodes
4. Require less than 2 GB of memory

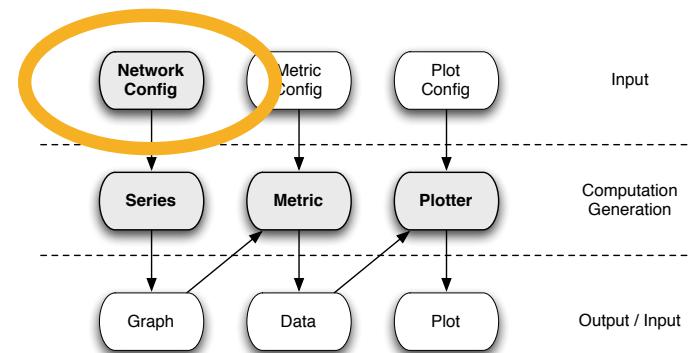
2. Modules and Workflow

The four Modules and how they build GTNA



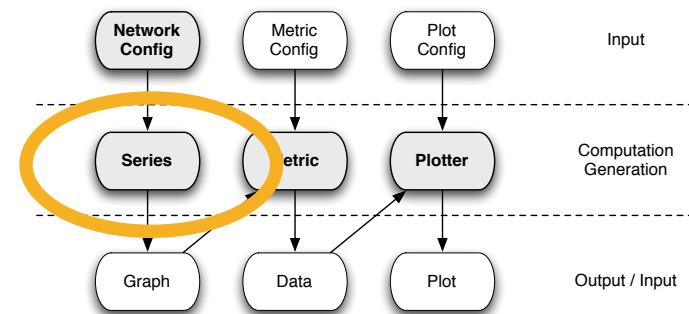
Network Config

- Interface: *gtna.network.Network*
 1. Set of parameters
 2. Constructor
 3. Graph generation
- E.g. CAN(nodes, dimensions, realities)



Series

- Implementation: *gtna.data.Series*
- Aggregates several simulation runs
 - Instances of the same network config
 - Averages + confidence intervals
- Input
 - a. Generated network topology
 - b. Imported available traces



Metric

- Interface: *gtna.metrics.Metric*

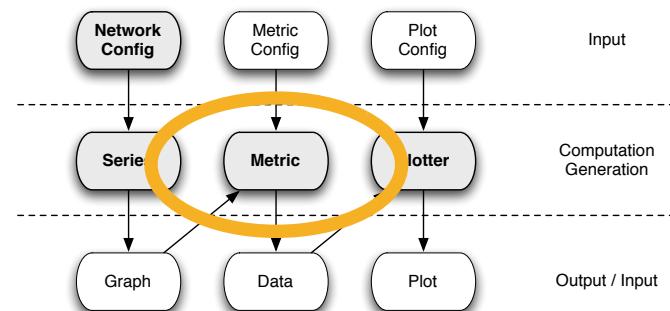
1. Sub-metric attributes
2. Constructor
3. Computation
4. Output generation

- Multi-scalar metrics

- Array of values for each graph

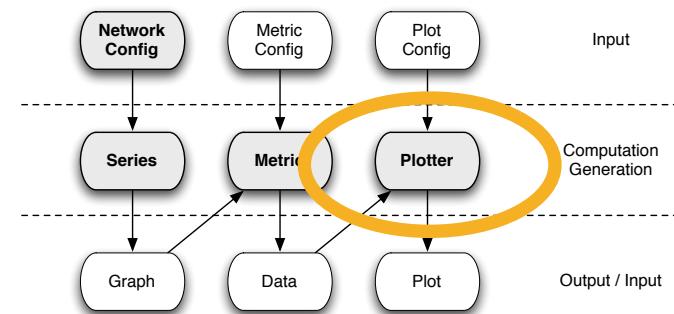
- Single-scalar metrics

- Single value for each graph

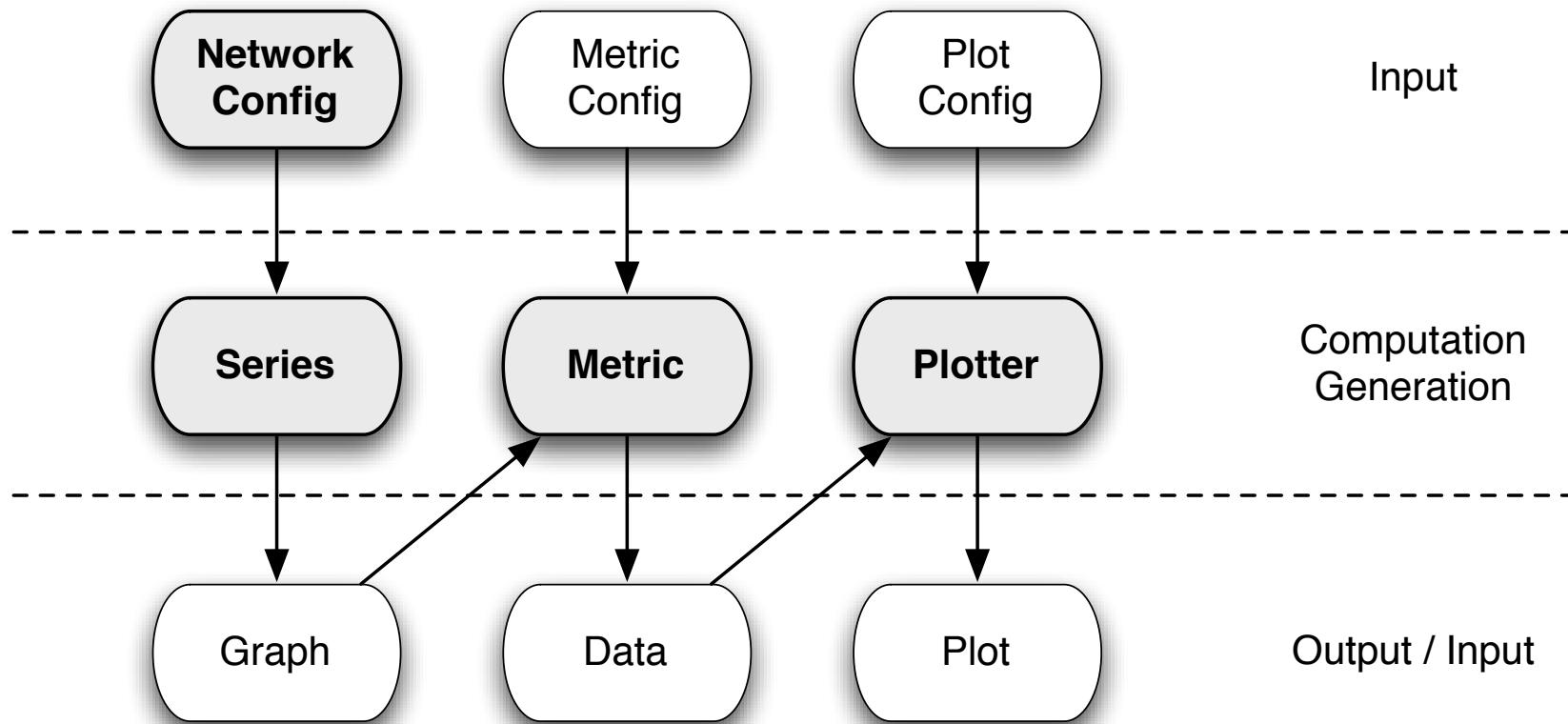


Plotter

- Implementation: *gtna.plot.Plot*
 - Uses *gtna.plot.GNUPlot*
- Plot types
 - Multi, Single, byEdges
- Combination of
 - Multiple metrics
 - Different configuration
 - Different networks



Workflow

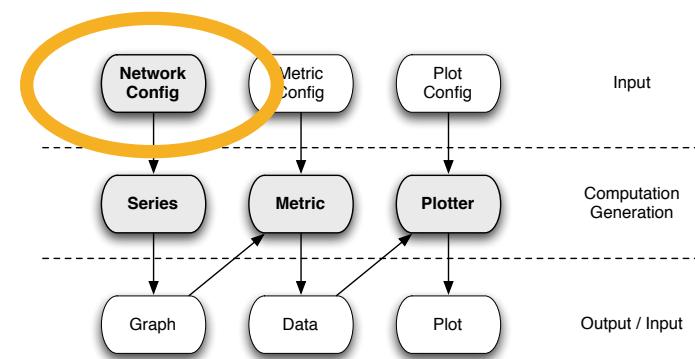


Instantiating a Network Config

```
Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[ ] can_x_1 =
    new Network[]{ can_2_1, can_3_1, can_4_1 };

Network can_2_1 = new CAN(100, 2, 1);
Network can_2_2 = new CAN(100, 2, 2);
Network can_2_3 = new CAN(100, 2, 3);
Network[ ] can_2_x =
    new Network[]{ can_2_1, can_2_2, can_2_3 };

Network can_100 = new CAN(100, 2, 1);
Network can_200 = new CAN(200, 2, 1);
Network can_300 = new CAN(300, 2, 1);
Network can_400 = new CAN(400, 2, 1);
Network[ ] can_x =
    new Network[]{ can_100, can_200, can_300, can_400 };
```

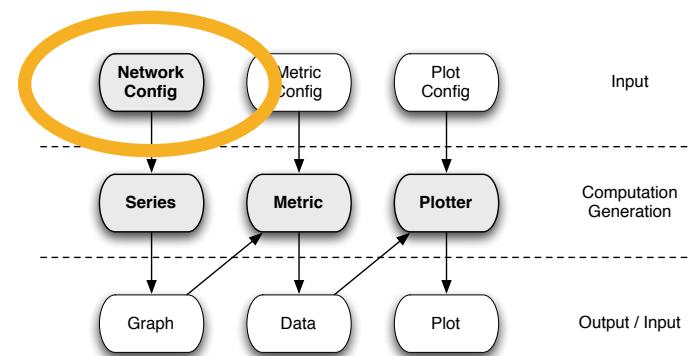


Instantiating a Network Config

```
Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[] can_x_1 =
    new Network[]{ can_2_1, can_3_1, can_4_1 };
```

```
Network can_2_1 = new CAN(100, 2, 1);
Network can_2_2 = new CAN(100, 2, 2);
Network can_2_3 = new CAN(100, 2, 3);
Network[] can_2_x =
    new Network[]{ can_2_1, can_2_2, can_2_3 };
```

```
Network can_100 = new CAN(100, 2, 1);
Network can_200 = new CAN(200, 2, 1);
Network can_300 = new CAN(300, 2, 1);
Network can_400 = new CAN(400, 2, 1);
Network[] can_x =
    new Network[]{ can_100, can_200, can_300, can_400 };
```

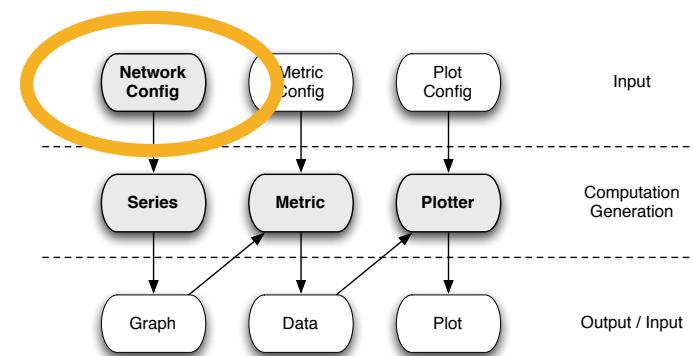


Instantiating a Network Config

```
Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[] can_x_1 =
    new Network[]{ can_2_1, can_3_1, can_4_1 };
```

```
Network can_2_1 = new CAN(100, 2, 1);
Network can_2_2 = new CAN(100, 2, 2);
Network can_2_3 = new CAN(100, 2, 3);
Network[] can_2_x =
    new Network[]{ can_2_1, can_2_2, can_2_3 };
```

```
Network can_100 = new CAN(100, 2, 1);
Network can_200 = new CAN(200, 2, 1);
Network can_300 = new CAN(300, 2, 1);
Network can_400 = new CAN(400, 2, 1);
Network[] can_x =
    new Network[]{ can_100, can_200, can_300, can_400 };
```



Instantiating a Network Config

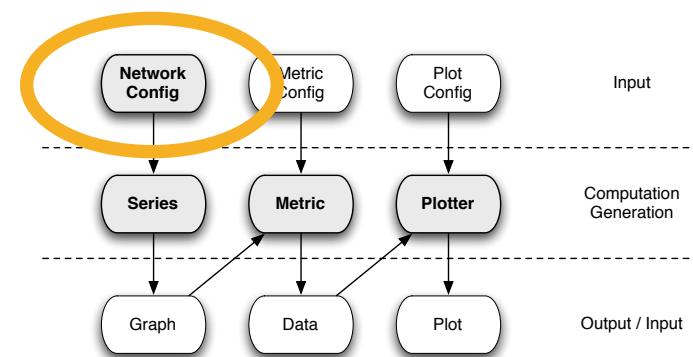
```

Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[] can_x_1 =
    new Network[]{ can_2_1, can_3_1, can_4_1 };

Network can_2_1 = new CAN(100, 2, 1);
Network can_2_2 = new CAN(100, 2, 2);
Network can_2_3 = new CAN(100, 2, 3);
Network[] can_2_x =
    new Network[]{ can_2_1, can_2_2, can_2_3 };

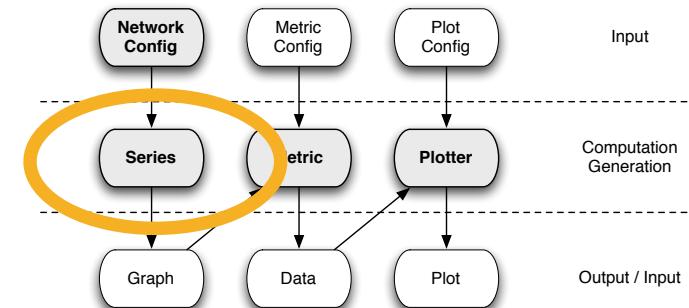
Network can_100 = new CAN(100, 2, 1);
Network can_200 = new CAN(200, 2, 1);
Network can_300 = new CAN(300, 2, 1);
Network can_400 = new CAN(400, 2, 1);
Network[] can_x =
    new Network[]{ can_100, can_200, can_300, can_400 };

```



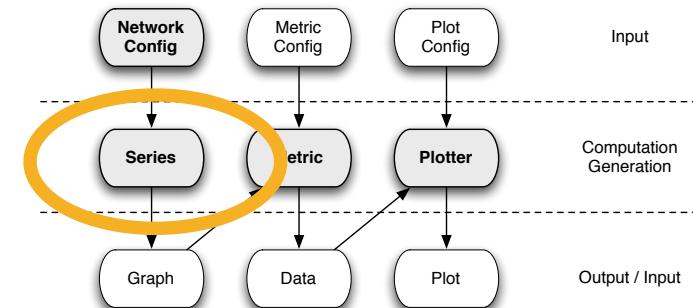
Generating a Series

```
Network[] can_x_1 =  
    new Network[]{ can_2_1, can_3_1, can_4_1 };  
  
Network[] can_2_x =  
    new Network[]{ can_2_1, can_2_2, can_2_3 };  
  
Network[] can_x =  
    new Network[]{ can_100, can_200, can_300, can_400 };  
  
Series[] s_can_x_1 = Series.generate(can_x_1, 20);  
Series[] s_can_2_x = Series.generate(can_2_x, 20);  
Series[] s_can_x = Series.generate(can_x, 20);
```



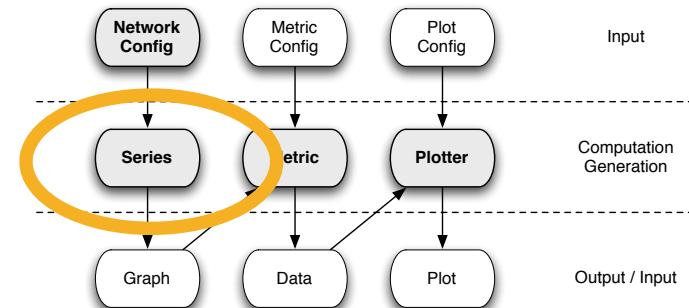
Generating a Series

```
Network[] can_x_1 =  
    new Network[]{ can_2_1, can_3_1, can_4_1 };  
  
Network[] can_2_x =  
    new Network[]{ can_2_1, can_2_2, can_2_3 };  
  
Network[] can_x =  
    new Network[]{ can_100, can_200, can_300, can_400 };  
  
Series[] s_can_x_1 = Series.generate(can_x_1, 20);  
Series[] s_can_2_x = Series.generate(can_2_x, 20);  
Series[] s_can_x = Series.generate(can_x, 20);
```

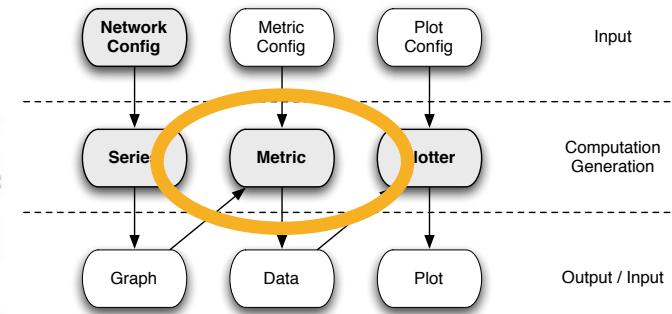
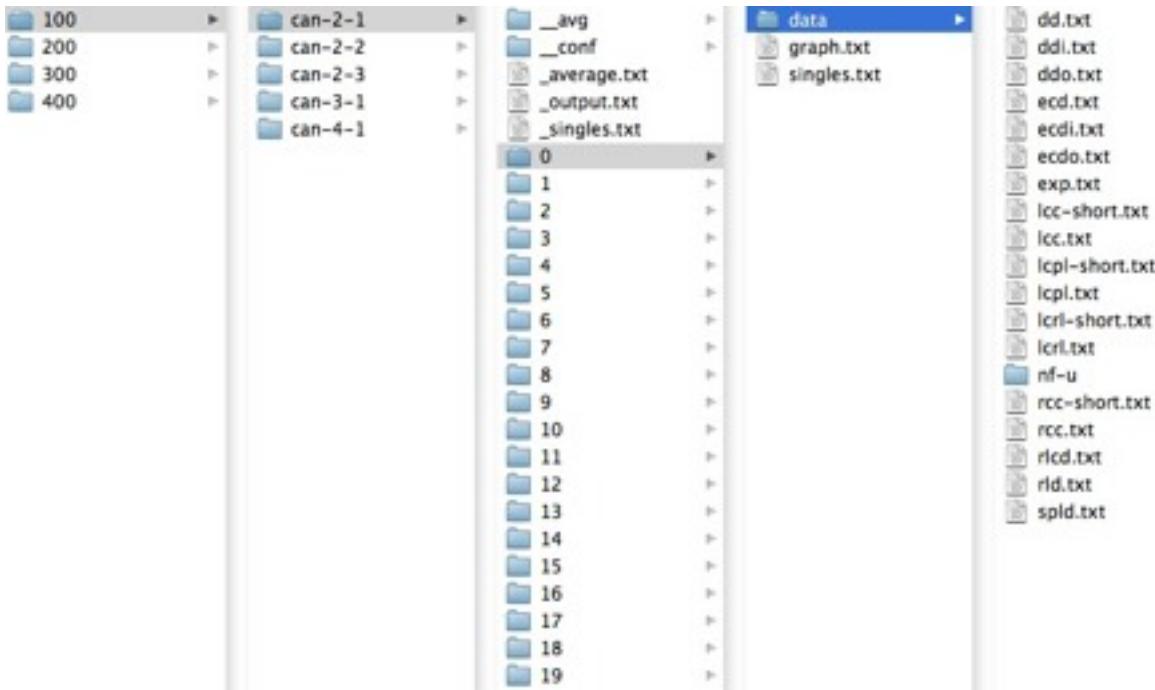


Generating a Series

```
Network[] can_x_1 =  
    new Network[]{ can_2_1, can_3_1, can_4_1 };  
  
Network[] can_2_x =  
    new Network[]{ can_2_1, can_2_2, can_2_3 };  
  
Network[] can_x =  
    new Network[]{ can_100, can_200, can_300, can_400 };  
  
Series[] s_can_x_1 = Series.generate(can_x_1, 20);  
Series[] s_can_2_x = Series.generate(can_2_x, 20);  
Series[] s_can_x = Series.generate(can_x, 20);
```



Data Generation



Plotting the Results

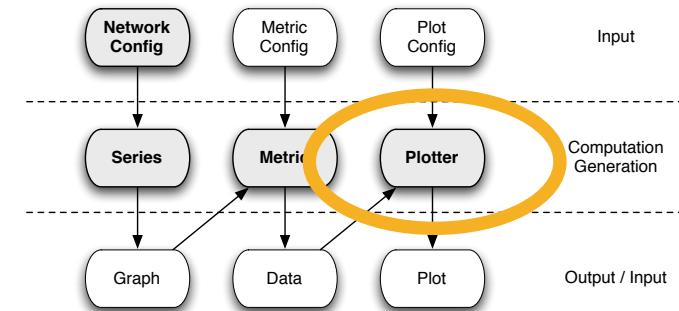
```
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");

Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x };

Plot.allSingleByEdges(s_all, "./can-by-edges/");
```



Plotting the Results

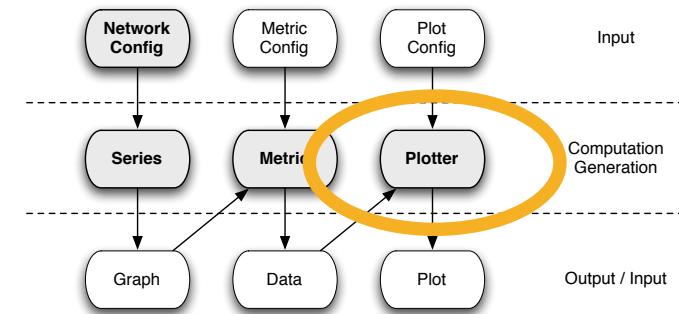
```
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");

Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x };

Plot.allSingleByEdges(s_all, "./can-by-edges/");
```



Plotting the Results

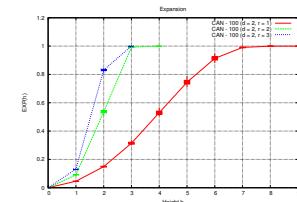
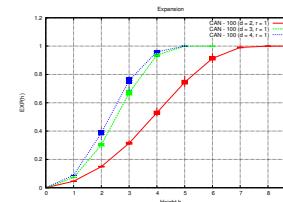
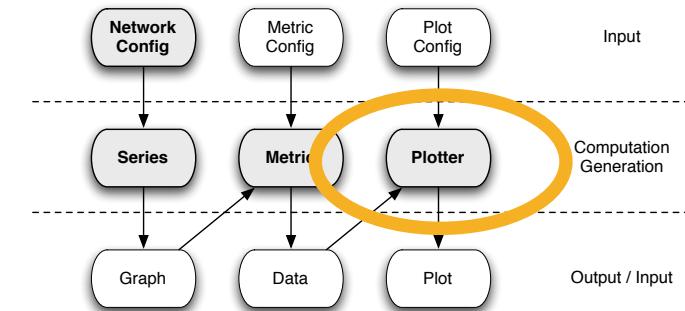
```
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");

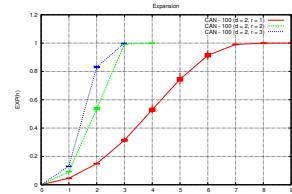
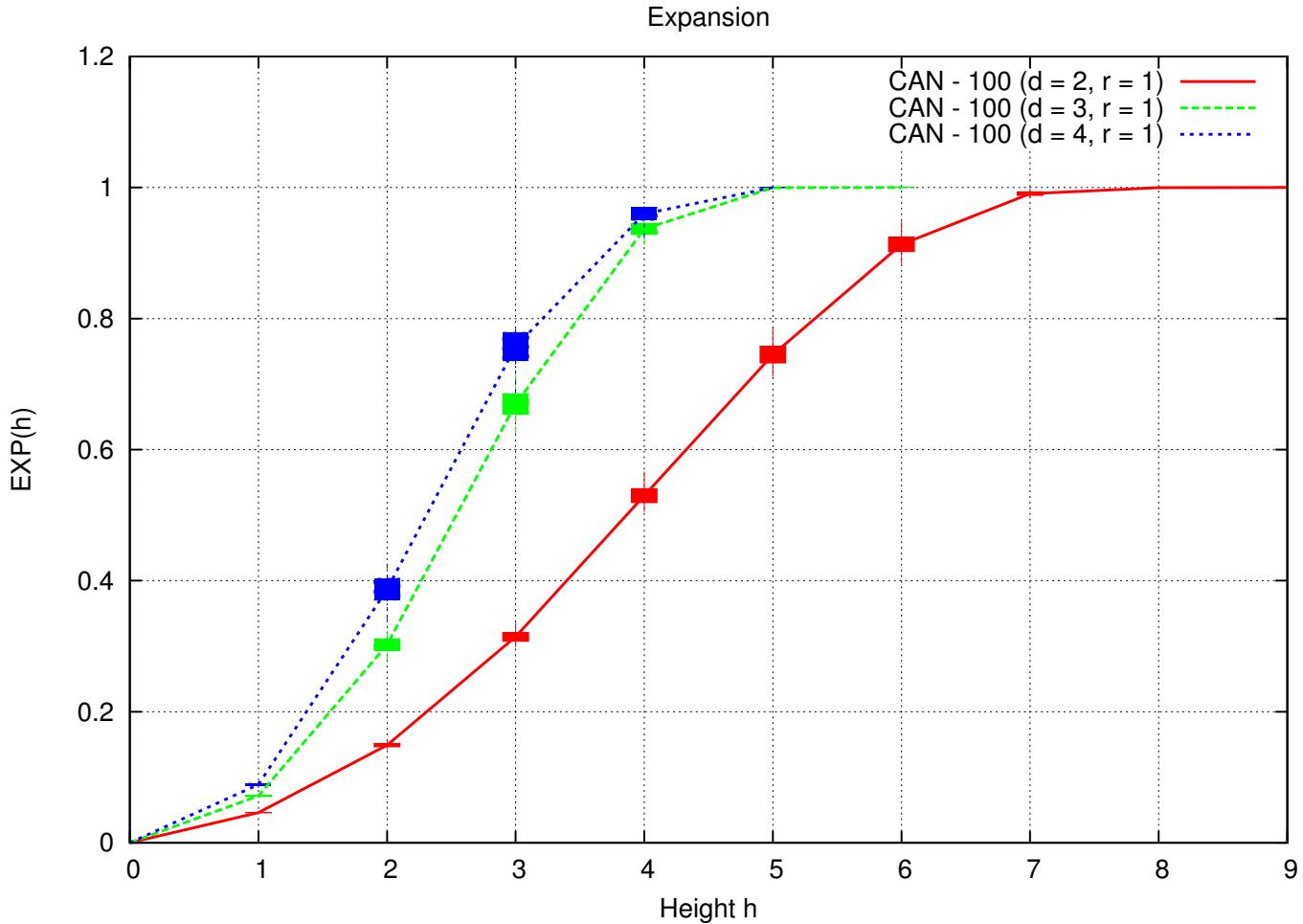
Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x };

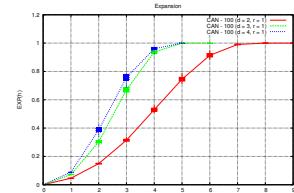
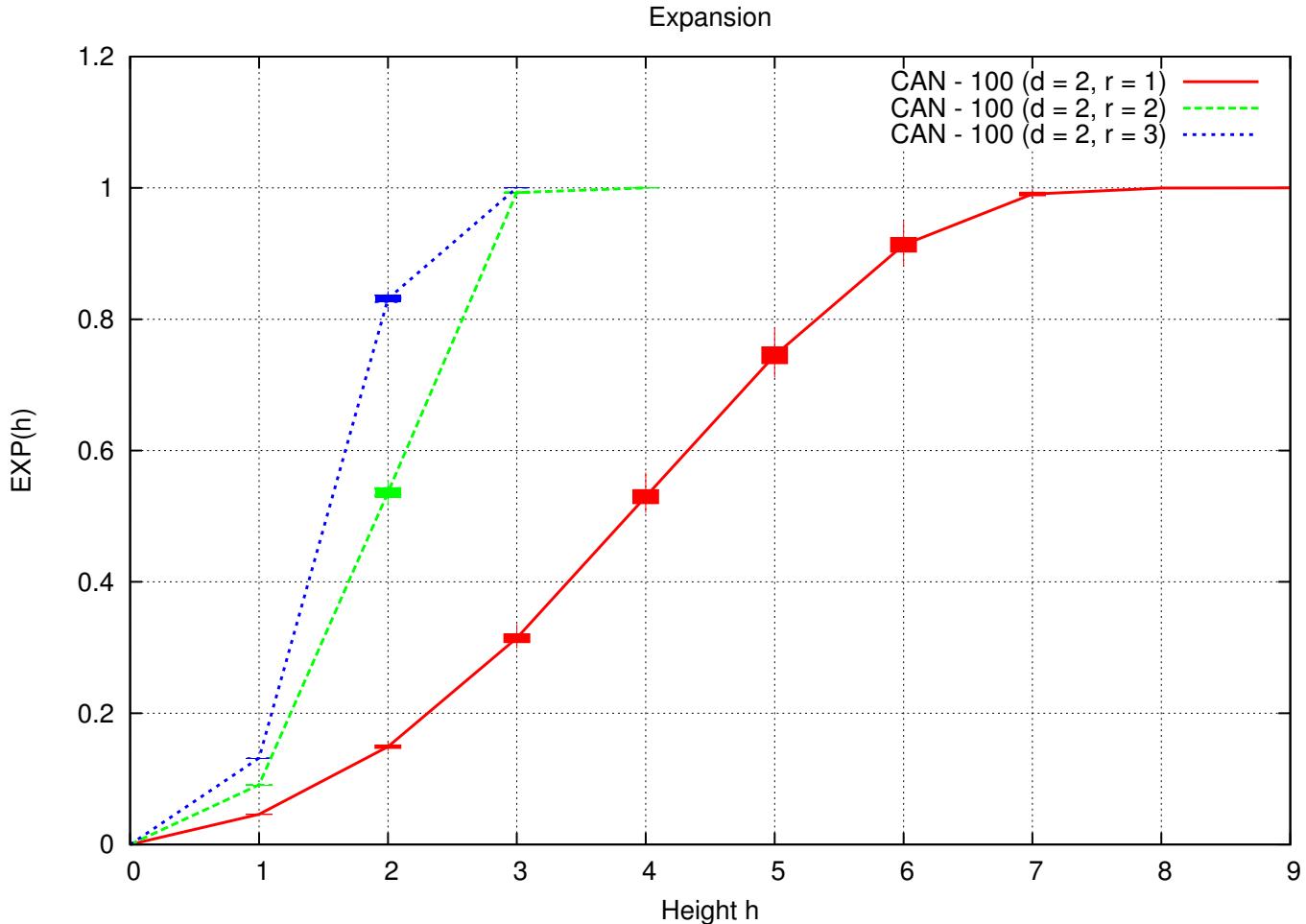
Plot.allSingleByEdges(s_all, "./can-by-edges/");
```



Plotting the Results



Plotting the Results



Plotting the Results



```

Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

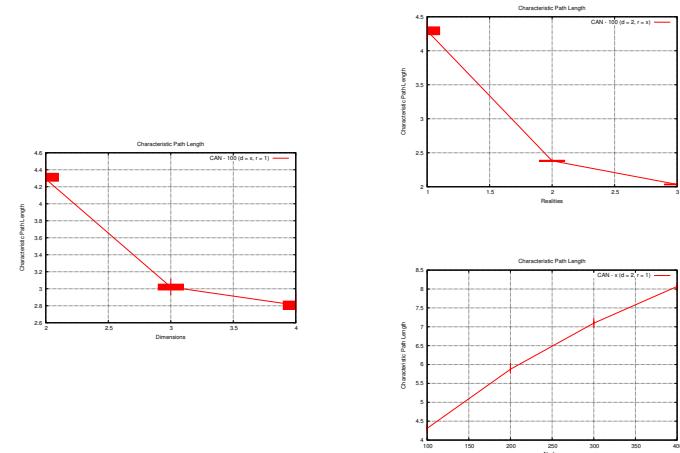
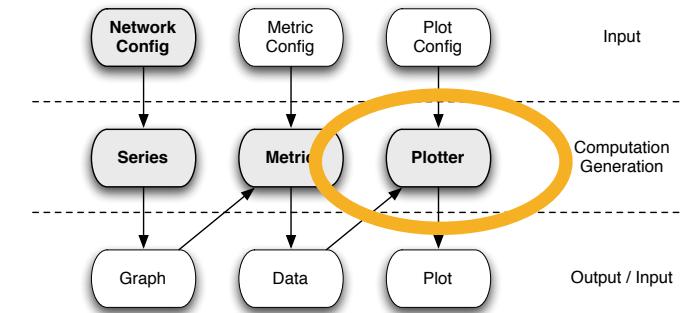
Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");

Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

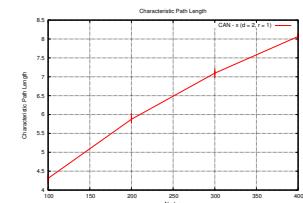
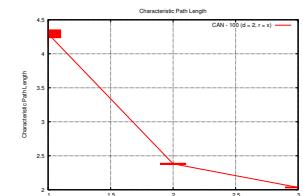
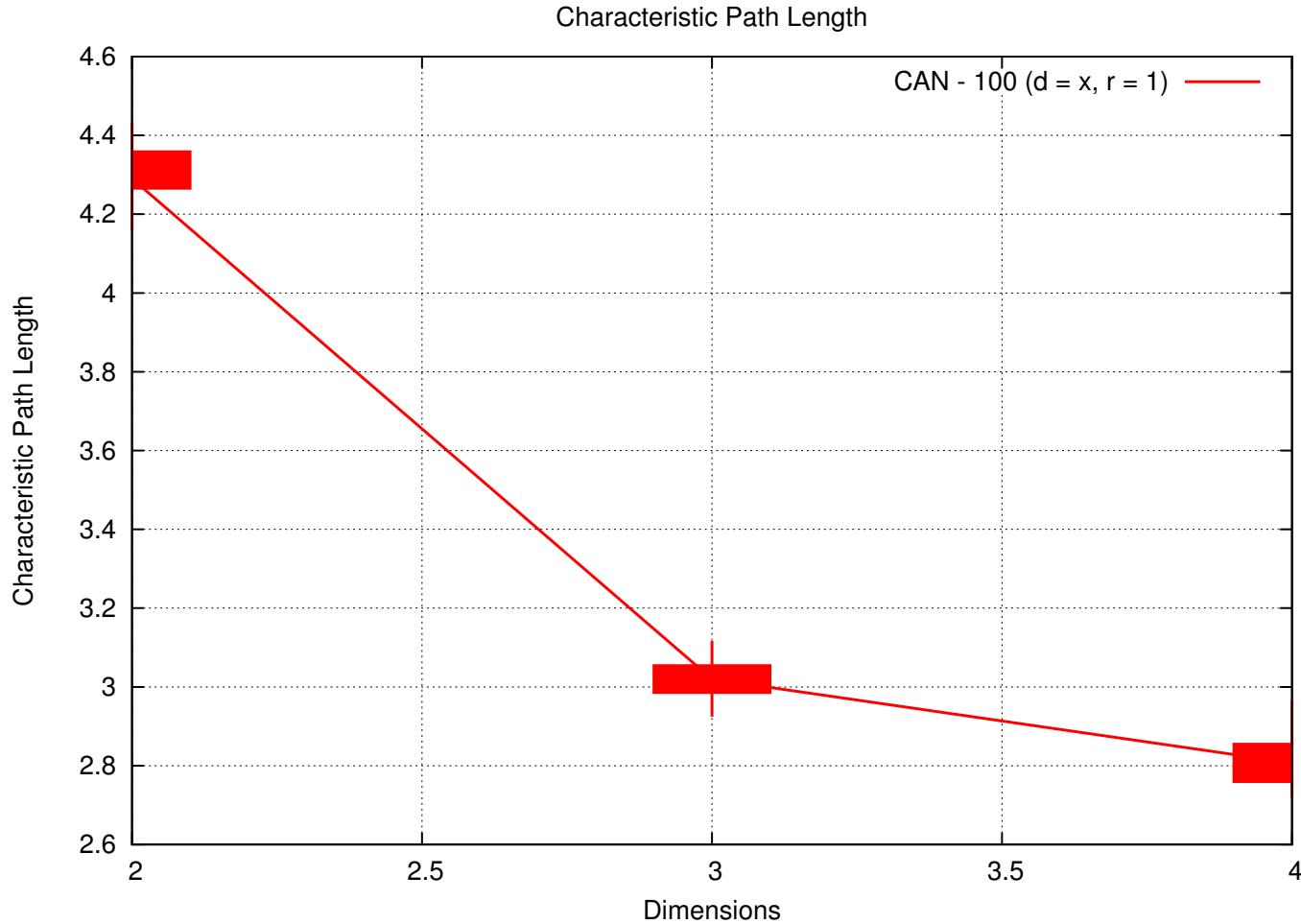
Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x };

Plot.allSingleByEdges(s_all, "./can-by-edges/");

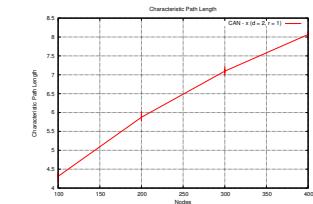
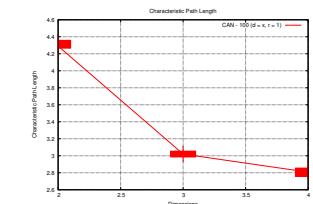
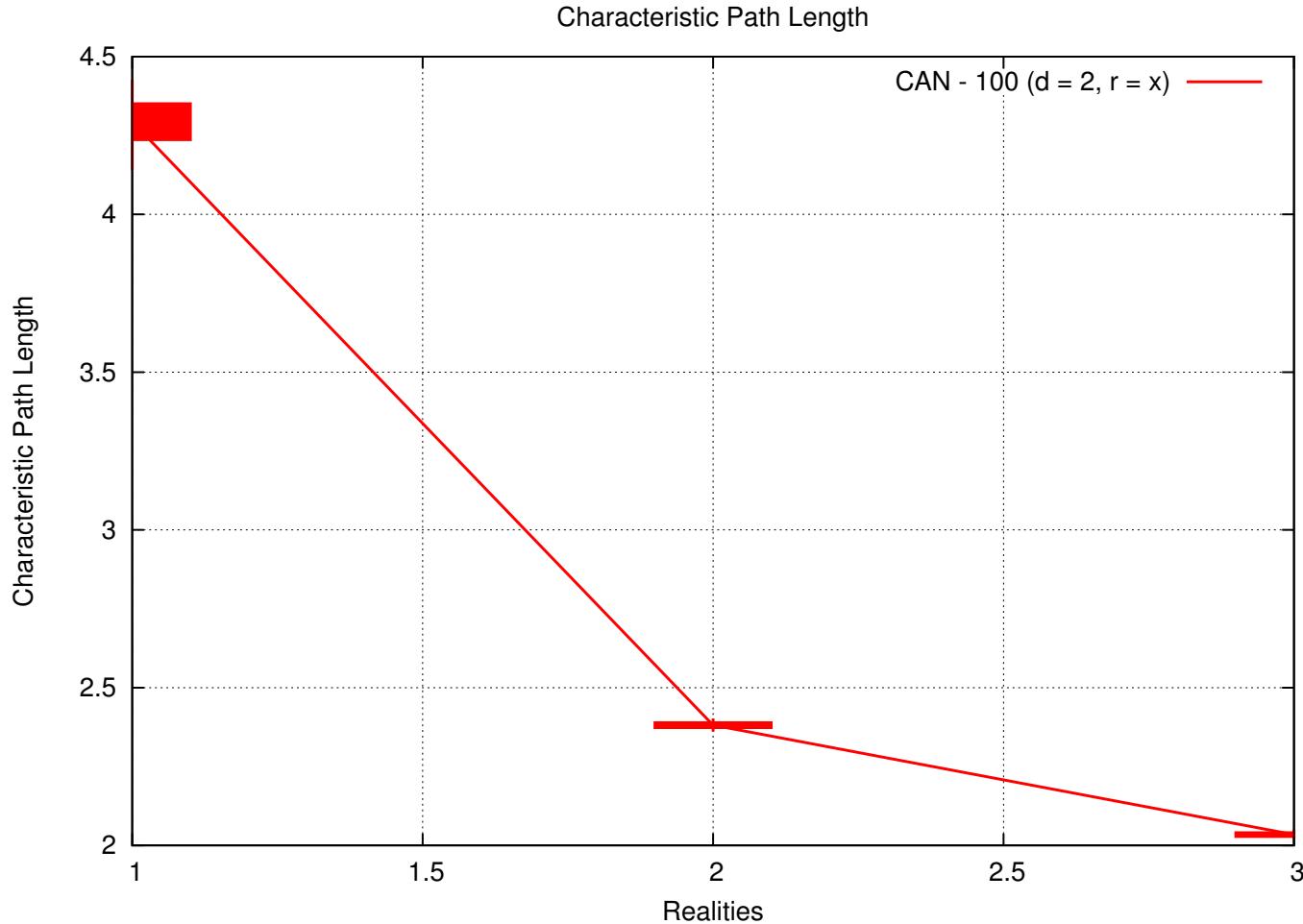
```



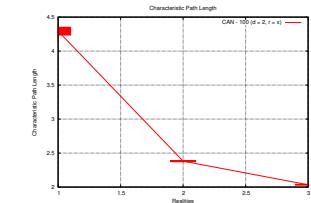
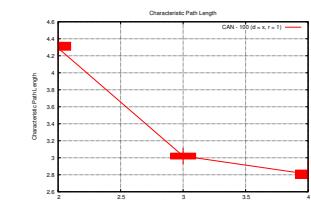
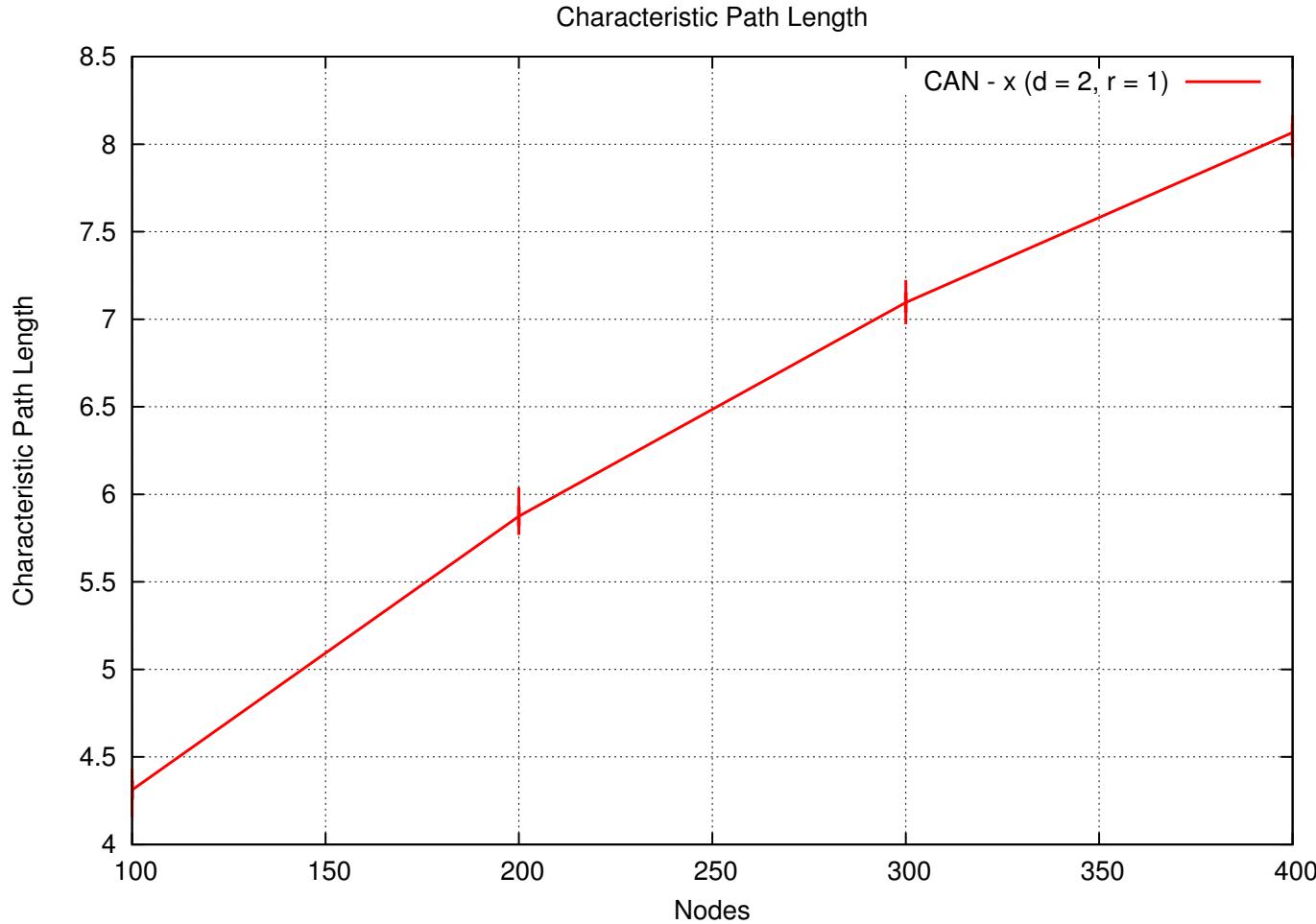
Plotting the Results



Plotting the Results



Plotting the Results



Plotting the Results

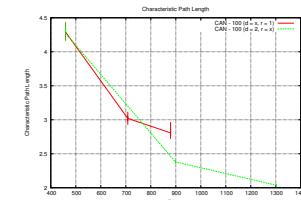
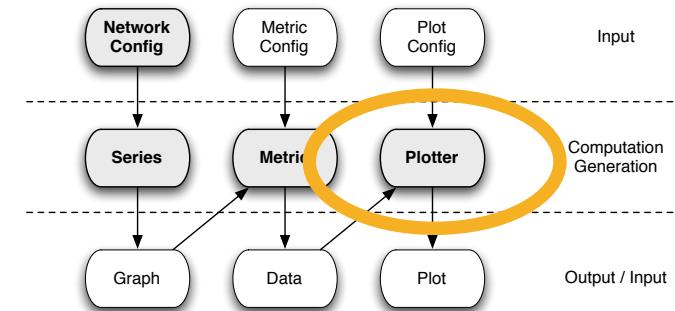
```
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");

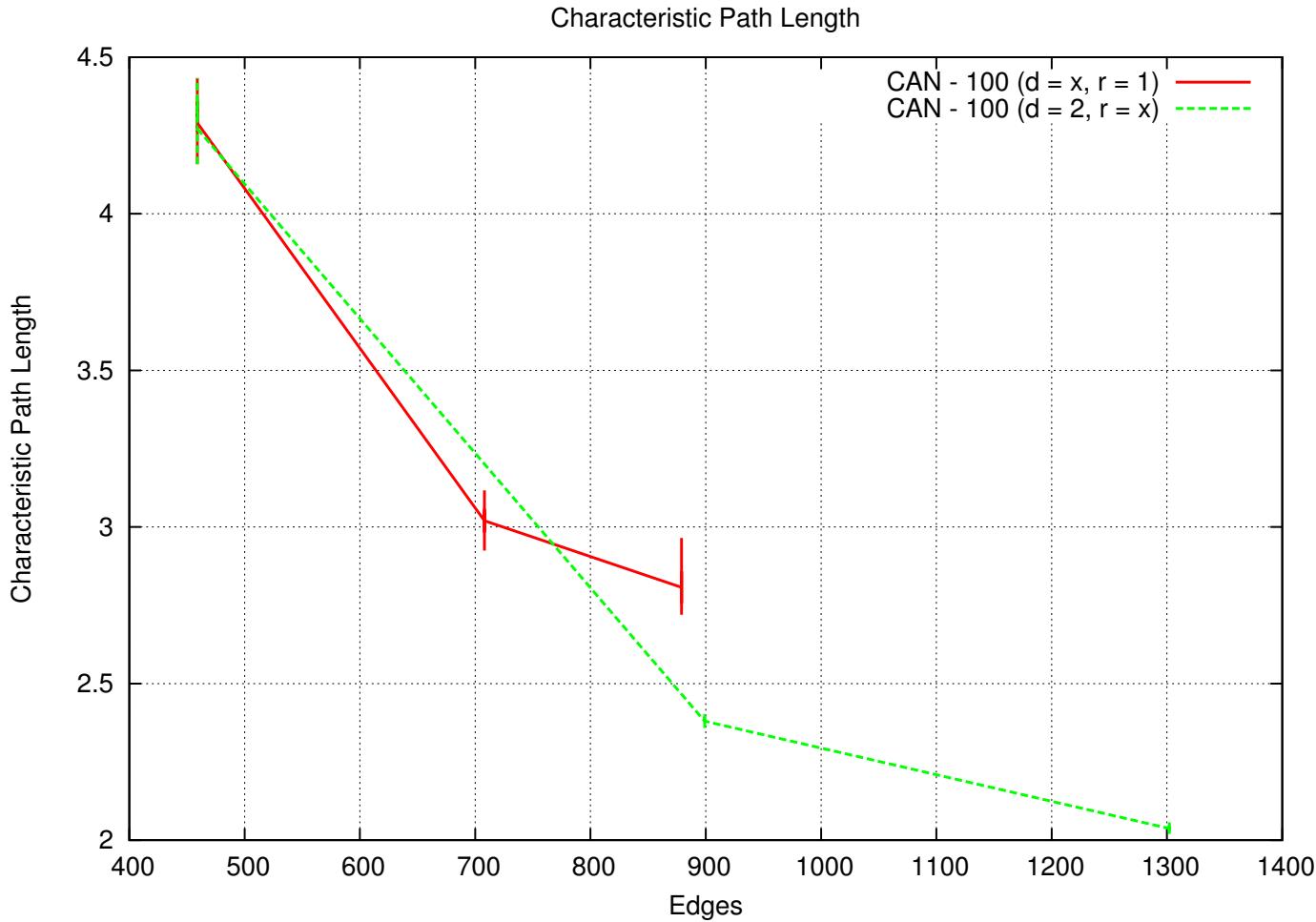
Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x };

Plot.allSingleByEdges(s_all, "./can-by-edges/");
```

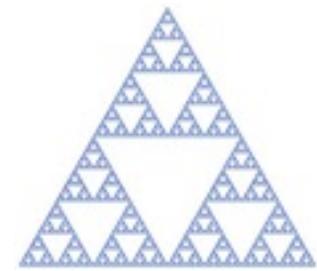
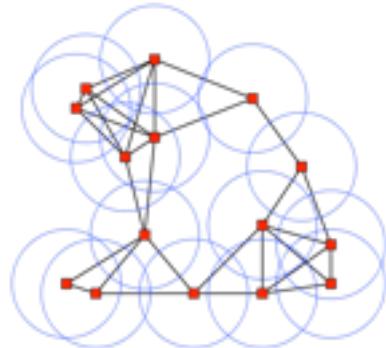
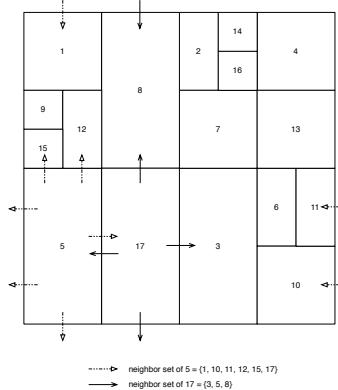
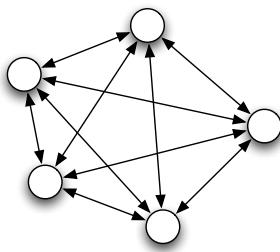


Plotting the Results



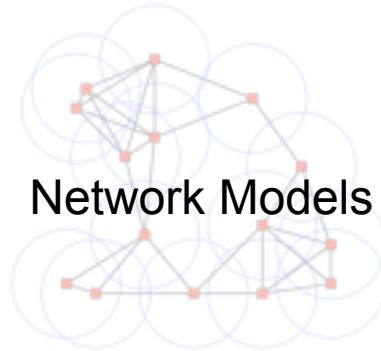
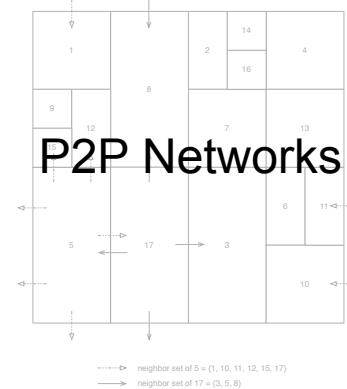
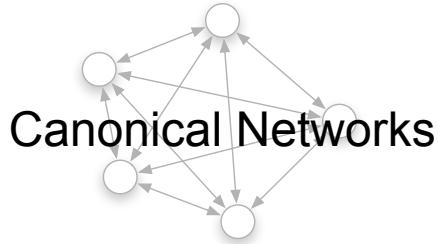
3. Available Networks

Networks already implemented in GTNA



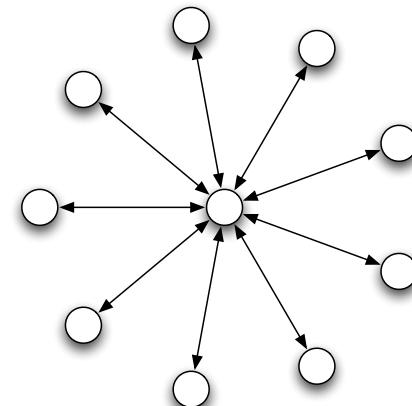
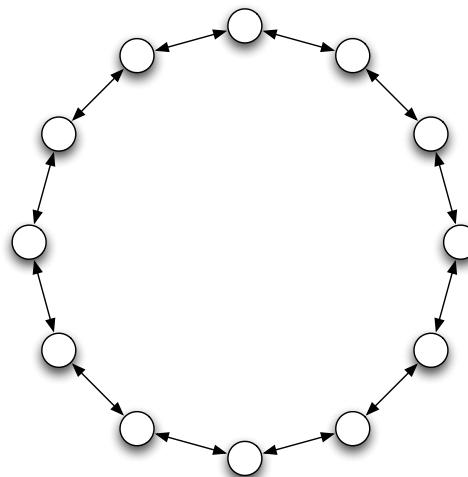
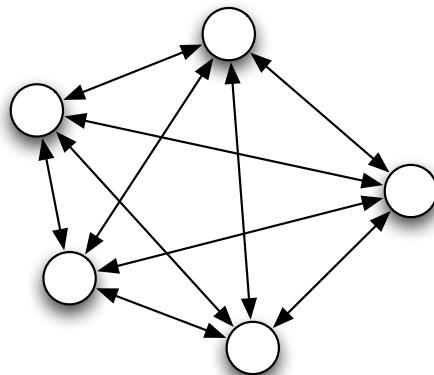
3. Available Networks

Networks already implemented in GTNA



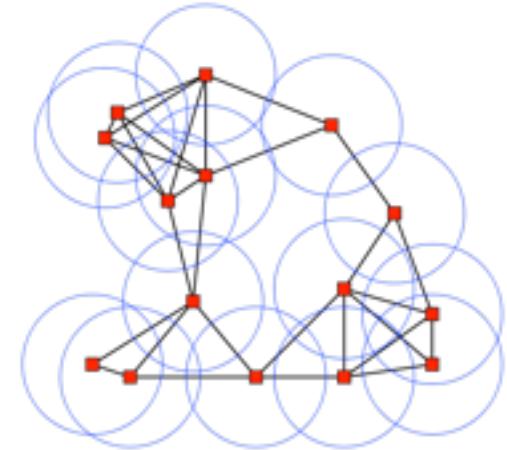
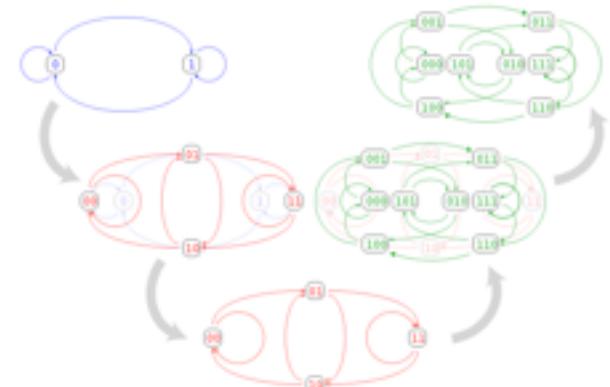
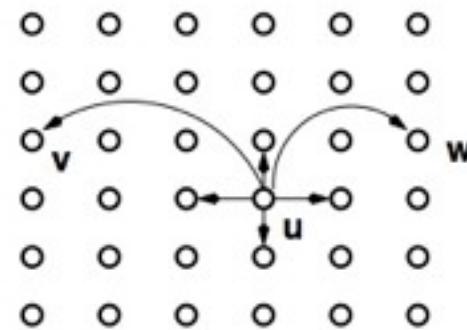
Canonical Networks

- Complete
- Ring
- Star



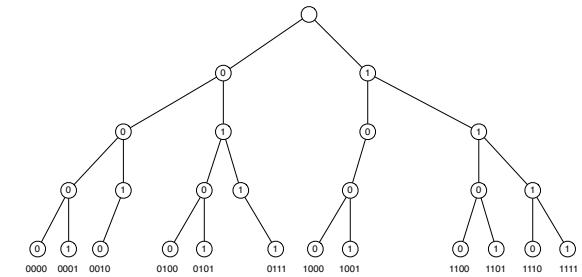
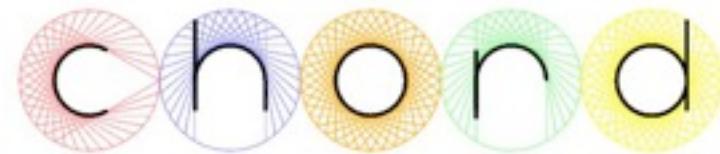
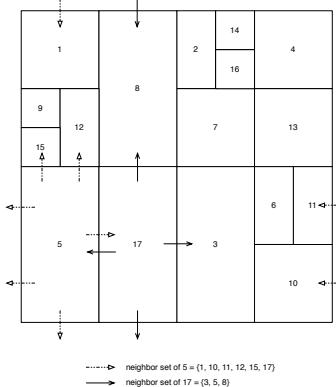
Network Models

- Barabasi Albert
- De Bruijn
- Erdos Renyi
- Gilbert
- GNC - Growing Network with Copying
- GNR - Growing Network with Re-direction
- Kleinberg
- UnitDisc
- Watts Strogatz



Peer-to-Peer Networks

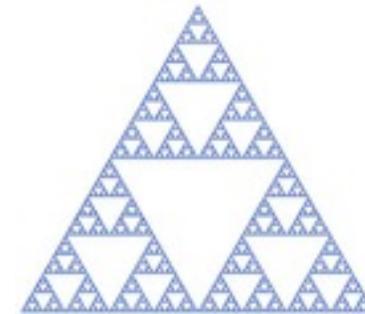
- CAN
- Chord
- Gnutella 0.4
- Gnutella 0.6
- Kademlia
- ODRI - Optimal Diameter Routing Infrastructure
- Pastry
- PathFinder
- Symphony



Misc Networks

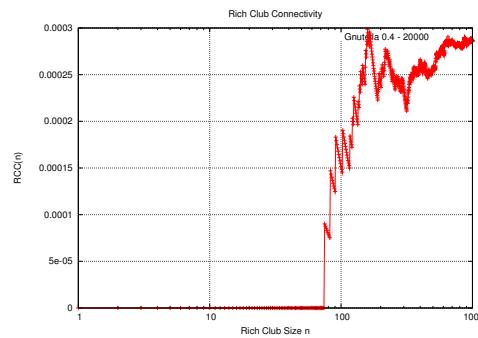
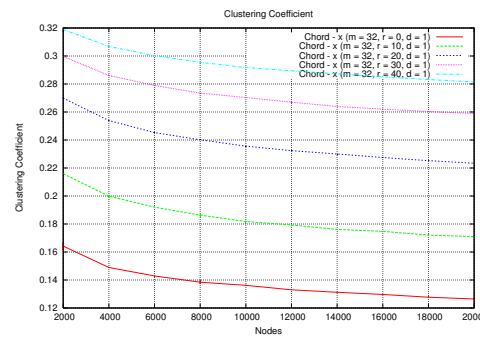
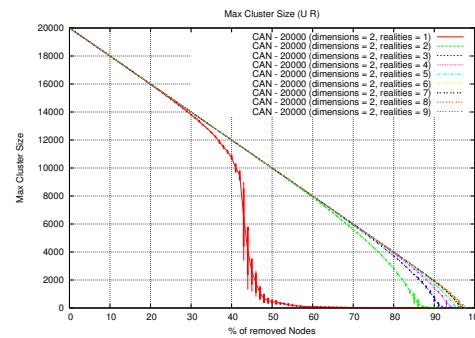
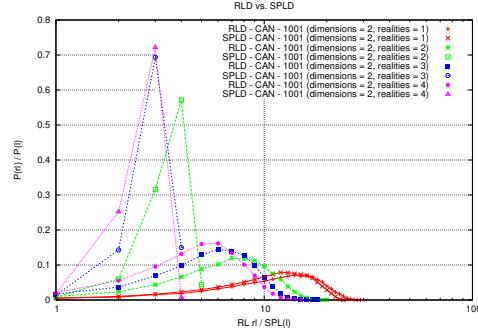
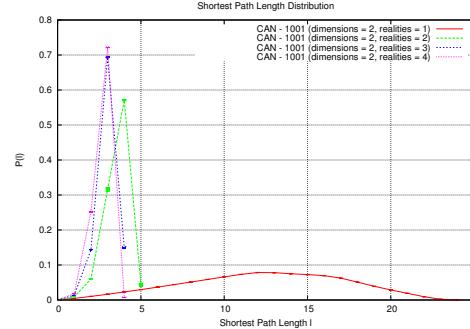
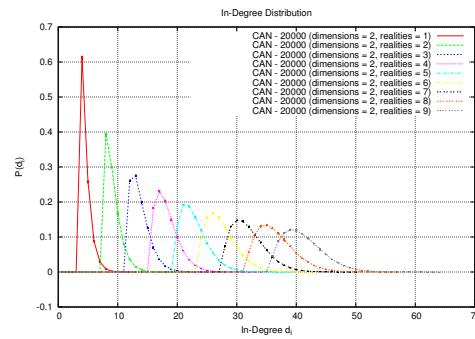
- Utilities
 - Readable File
 - Readable List
- Transformations
 - Add Edges Per Node
 - Add Edges Randomly
 - Rewire
 - Same Degree Distribution

1. own format
2. edges only
3. GML
4. caida.org Internet traces



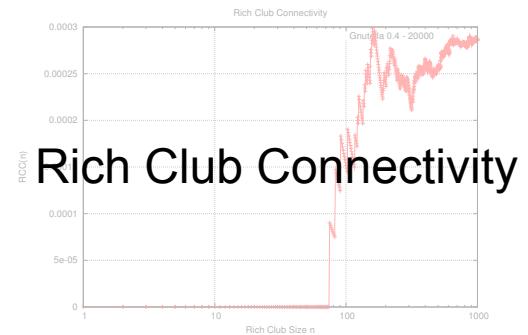
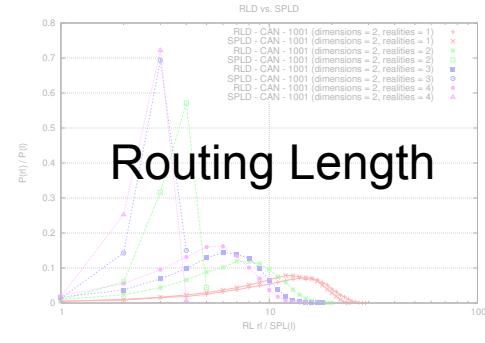
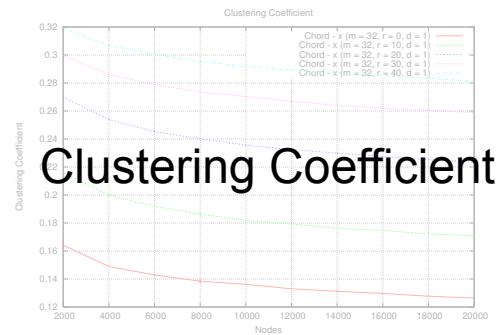
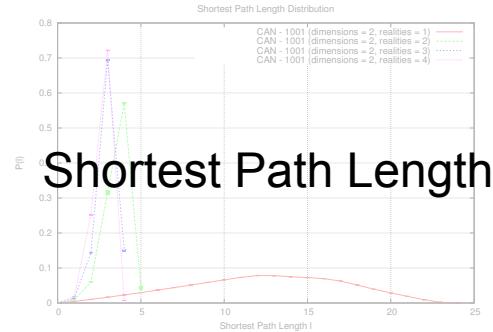
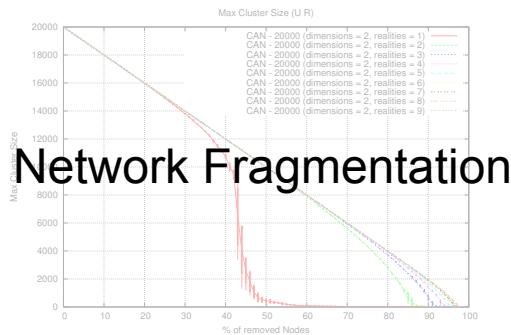
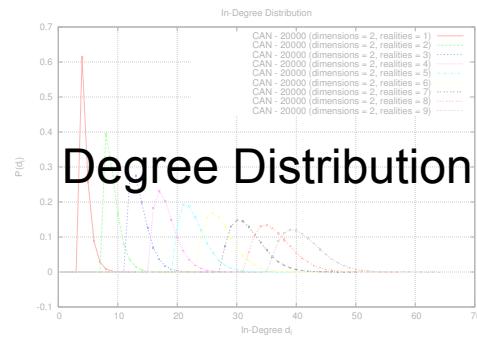
4. Available Metrics

Metrics already implemented in GTNA



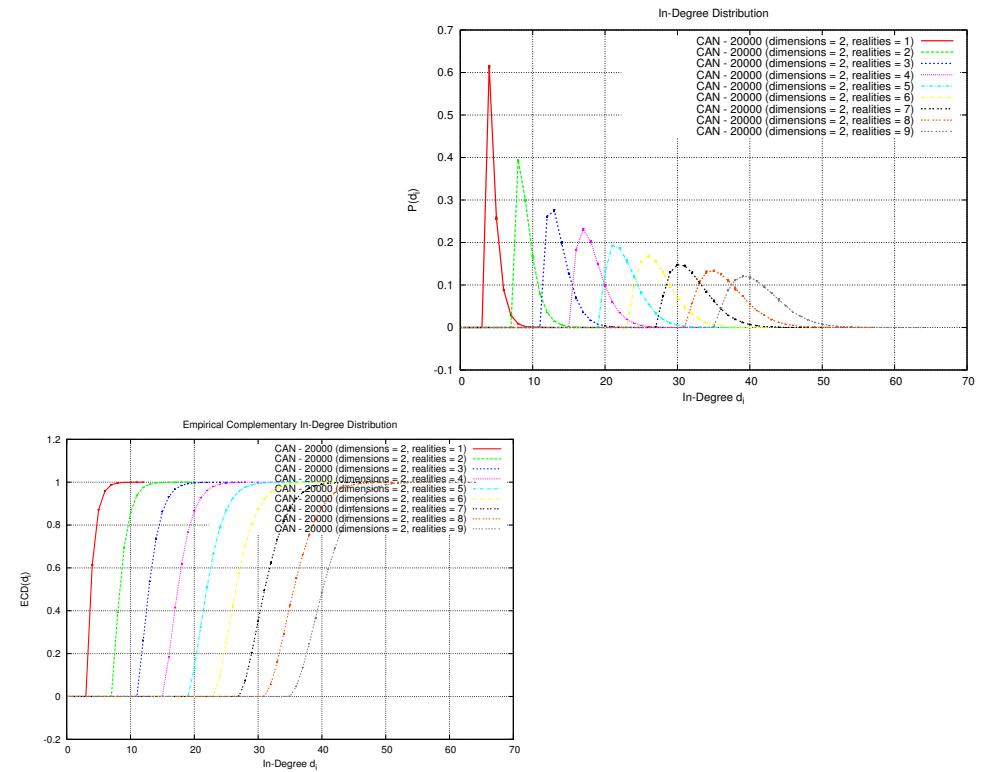
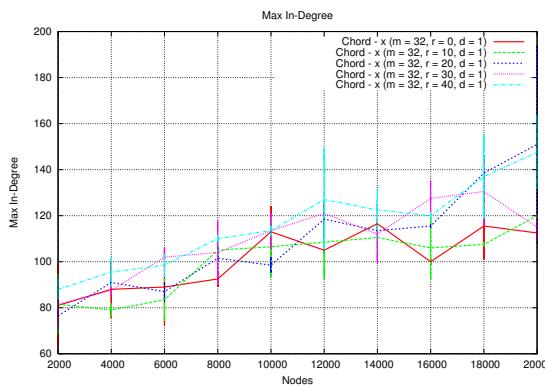
4. Available Metrics

Metrics already implemented in GTNA



Degree Distribution

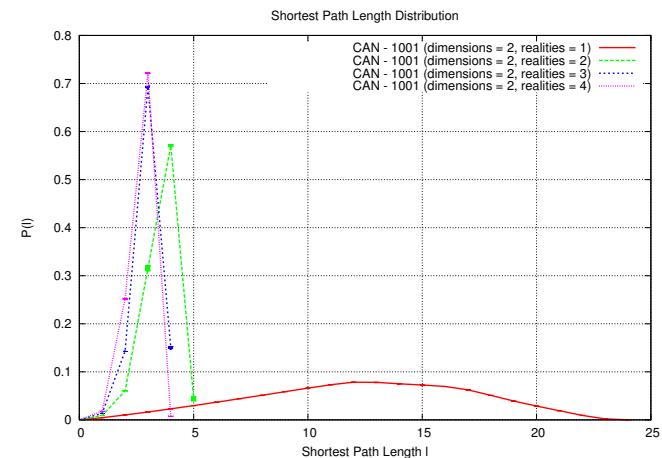
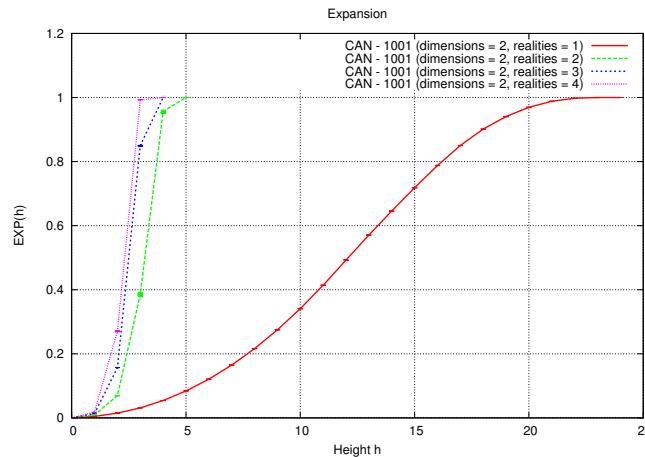
- (in-/out-)Degree Distribution
- Empirical Complementary (in-/out-)Degree Distribution
- #Nodes, #Edges
- Average Degree
- Min (in-/out-)Degree
- Max (in-/out-)Degree



Shortest Path Length



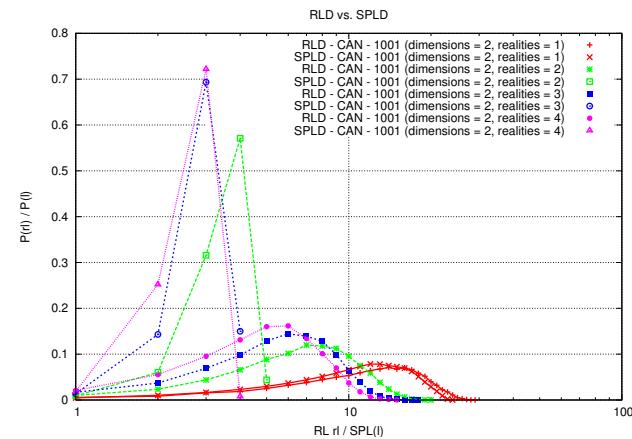
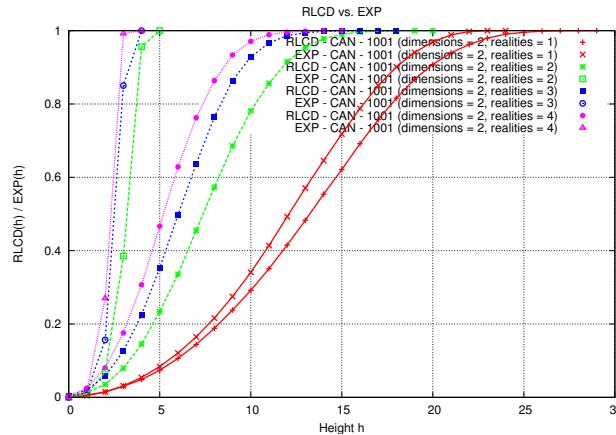
- Shortest Path Length Distribution
- Expansion
- Local Characteristic Path Length
- Characteristic Path Length
- Diameter



Routing Length



- Routing Length Distribution
- Routing Length Complementary Distribution
- Local Characteristic Routing Length
- Characteristic Routing Length
- Max Routing Length

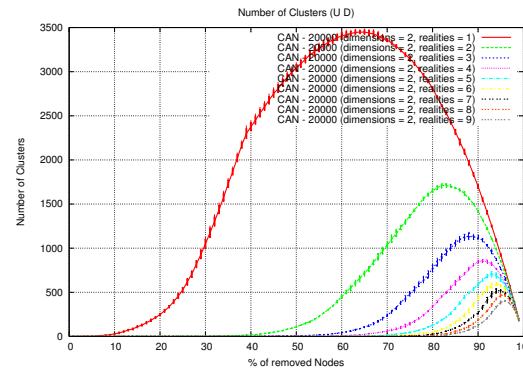
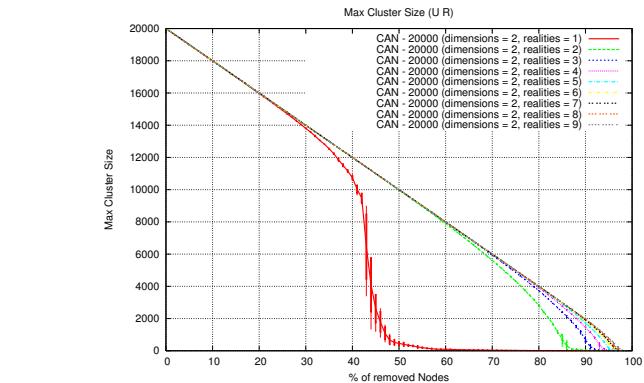
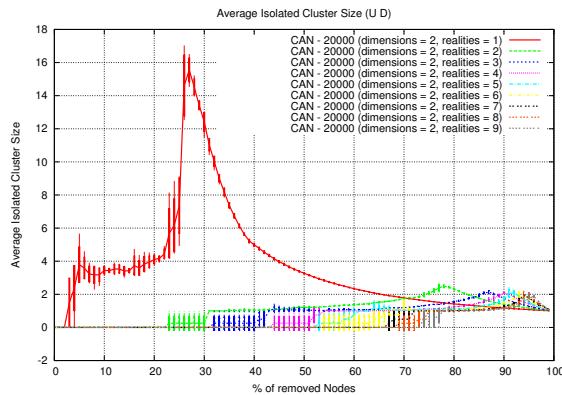


Network Fragmentation



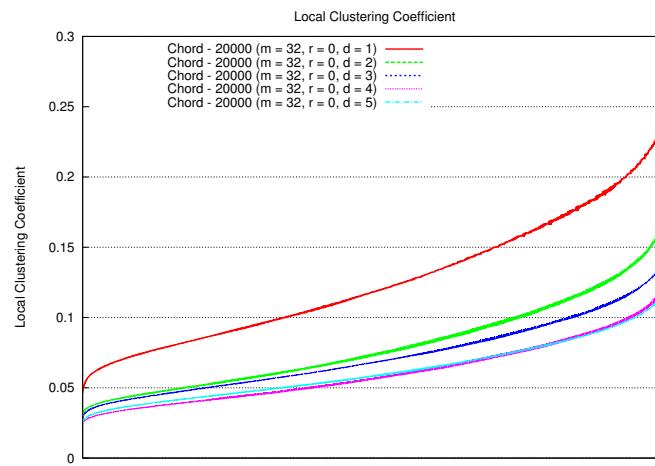
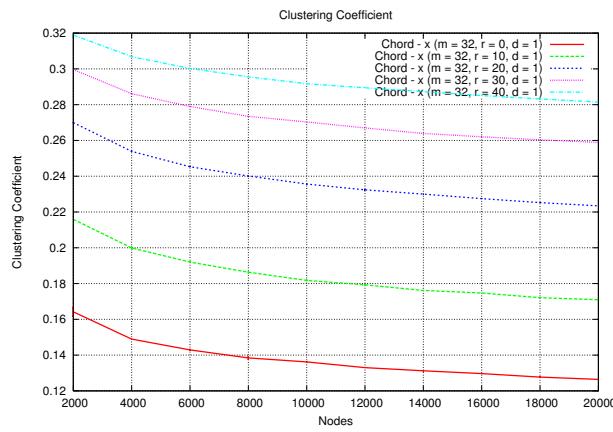
- Max Cluster Size
- Number of Clusters
- Average Isolated Cluster Size

- Point of Rupture
- Average Number of Clusters
- Max Number of Clusters



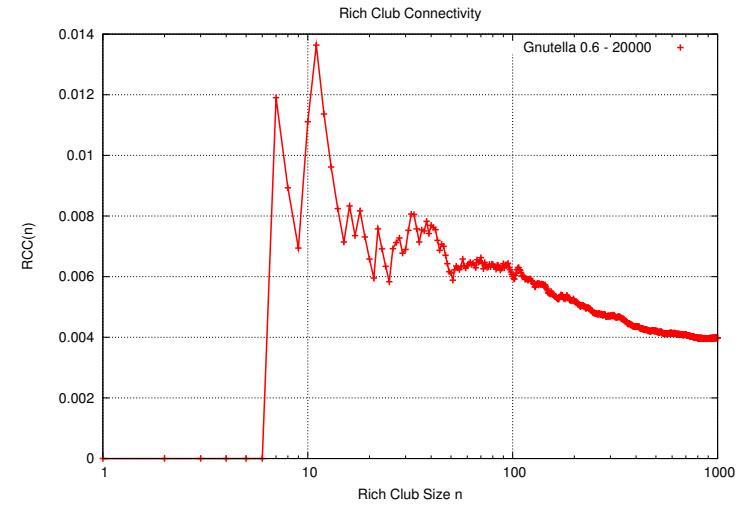
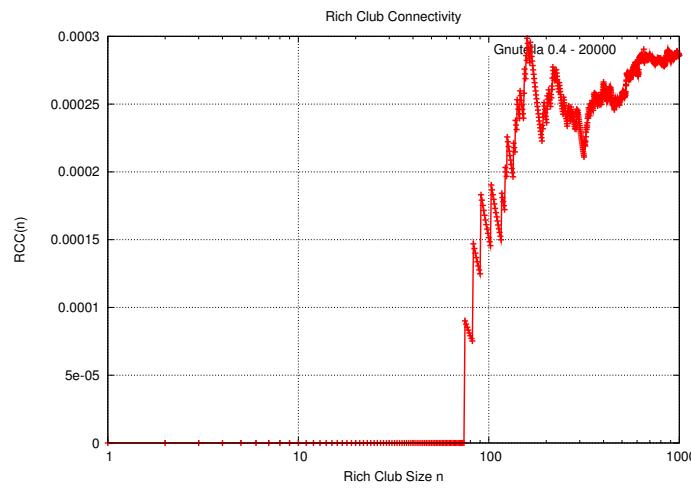
Clustering Coefficient

- Local Clustering Coefficient
- Clustering Coefficient
- Weighted Clustering Coefficient



Rich Club Connectivity

- Rich Club Connectivity



5. Extension



Implementation and Configuration of new Networks / Metrics

```
public class MyNetwork extends NetworkImpl implements Network {  
    private boolean p1;  
    public MyNetwork(int nodes, boolean p1) {  
        super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});  
        this.p1 = p1;  
    }  
    public Graph generate(){  
        Timer timer = new Timer();  
        Node[] nodes = Node.init(this.nodes());  
        Edges edges = new Edges(nodes, 100);  
        for(int i=1; i<nodes.length; i++){  
            edges.add(nodes[0], nodes[i]);  
            if(this.p1)  
                edges.add(nodes[i], nodes[0]);  
        }  
        edges.fill();  
        return new Graph(this.description(), nodes, timer);  
    }  
}
```

+

```
NOD_CLASS = gtna.metrics.NodesOfDegree  
NOD_NAME = Nodes of Degree  
NOD_DATA_KEYS = NOD_NOD  
NOD_SINGLES_KEYS = NOD_DOMN  
NOD_DATA_PLOTS = NOD  
NOD_SINGLES_PLOTS = DOMN  
  
NOD_NOD_DATA_NAME = Nodes of Degree  
NOD_NOD_FILENAME = nod  
  
NOD_DOMN_SINGLE_NAME = Degree of Most Nodes  
  
NOD_PLOT_DATA = NOD_NOD  
NOD_PLOT_FILENAME = nod  
NOD_PLOT_TITLE = Nodes of Degree  
NOD_PLOT_X = degree d  
NOD_PLOT_Y = # nodes of degree d  
  
DOMN_PLOT_DATA = NOD_DOMN  
DOMN_PLOT_FILENAME = domn  
DOMN_PLOT_TITLE = Degree of Most Nodes  
DOMN_PLOT_Y = Degree of Most Nodes
```

Implementing a new Network

```
public class MyNetwork extends NetworkImpl implements Network {  
    private boolean p1;  
    public MyNetwork(int nodes, boolean p1) {  
        super("MY_NETWORK", nodes, new String[] {"P1"}, new String[] {"" + p1});  
        this.p1 = p1;  
    }  
    public Graph generate(){  
        Timer timer = new Timer();  
        Node[] nodes = Node.init(this.nodes());  
        Edges edges = new Edges(nodes, 100);  
        for(int i=1; i<nodes.length; i++){  
            edges.add(nodes[i], nodes[0]);  
            if(this.p1)  
                edges.add(nodes[0], nodes[i]);  
        }  
        edges.fill();  
        return new Graph(this.description(), nodes, timer);  
    }  
}
```

Implementing a new Network

```
public class MyNetwork extends NetworkImpl implements Network {  
    private boolean p1;  
    public MyNetwork(int nodes, boolean p1) {  
        super("MY_NETWORK", nodes, new String[] {"P1"}, new String[] {"" + p1});  
        this.p1 = p1;  
    }  
    public Graph generate(){  
        Timer timer = new Timer();  
        Node[] nodes = Node.init(this.nodes());  
        Edges edges = new Edges(nodes, 100);  
        for(int i=1; i<nodes.length; i++){  
            edges.add(nodes[i], nodes[0]);  
            if(this.p1)  
                edges.add(nodes[0], nodes[i]);  
        }  
        edges.fill();  
        return new Graph(this.description(), nodes, timer);  
    }  
}
```

Implementing a new Network

```
public class MyNetwork extends NetworkImpl implements Network {  
    private boolean p1;  
    public MyNetwork(int nodes, boolean p1) {  
        super("MY_NETWORK", nodes, new String[] {"P1"}, new String[] {"" + p1});  
        this.p1 = p1;  
    }  
    public Graph generate(){  
        Timer timer = new Timer();  
        Node[] nodes = Node.init(this.nodes());  
        Edges edges = new Edges(nodes, 100);  
        for(int i=1; i<nodes.length; i++){  
            edges.add(nodes[i], nodes[0]);  
            if(this.p1)  
                edges.add(nodes[0], nodes[i]);  
        }  
        edges.fill();  
        return new Graph(this.description(), nodes, timer);  
    }  
}
```

Parameter

Implementing a new Network

```
public class MyNetwork extends NetworkImpl implements Network {  
    private boolean p1;  
public MyNetwork(int nodes, boolean p1) {  
    super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});  
    this.p1 = p1;  
}  
public Graph generate(){  
    Timer timer = new Timer();  
    Node[] nodes = Node.init(this.nodes());  
    Edges edges = new Edges(nodes, 100);  
    for(int i=1; i<nodes.length; i++){  
        edges.add(nodes[i], nodes[0]);  
        if(this.p1)  
            edges.add(nodes[0], nodes[i]);  
    }  
    edges.fill();  
    return new Graph(this.description(), nodes, timer);  
}
```

Parameter

Constructor

Implementing a new Network

```
public class MyNetwork extends NetworkImpl implements Network {  
    private boolean p1;  
    public MyNetwork(int nodes, boolean p1) {  
        super("MY_NETWORK", nodes, new String[] {"P1"}, new String[] {" " + p1});  
        this.p1 = p1;  
    }  
    public Graph generate(){  
        Timer timer = new Timer();  
        Node[] nodes = Node.init(this.nodes());  
        Edges edges = new Edges(nodes, 100);  
        for(int i=1; i<nodes.length; i++){  
            edges.add(nodes[i], nodes[0]);  
            if(this.p1)  
                edges.add(nodes[0], nodes[i]);  
        }  
        edges.fill();  
        return new Graph(this.description(), nodes, timer);  
    }  
}
```

Parameter

Constructor

Generator

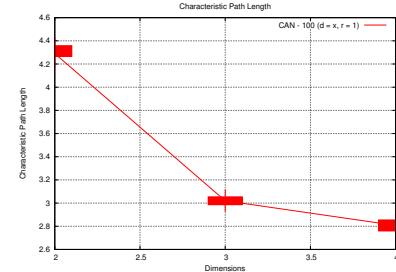
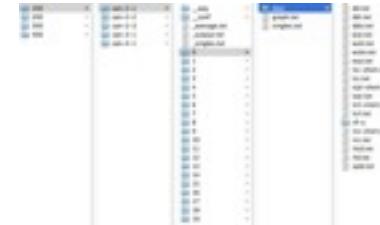
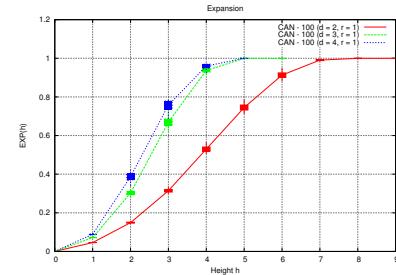
Configuring a new Network

```
CAN_NETWORK_NAME = CAN
```

```
CAN_NETWORK_FOLDER = can
```

```
CAN_NETWORK_DIMENSIONS_NAME = Dimensions
CAN_NETWORK_DIMENSIONS_NAME_LONG = dimensions
CAN_NETWORK_DIMENSIONS_NAME_SHORT = d
```

```
CAN_NETWORK_REALITIES_NAME = Realities
CAN_NETWORK_REALITIES_NAME_LONG = realities
CAN_NETWORK_REALITIES_NAME_SHORT = r
```



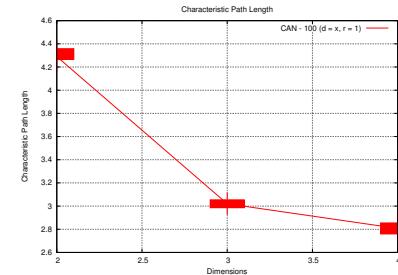
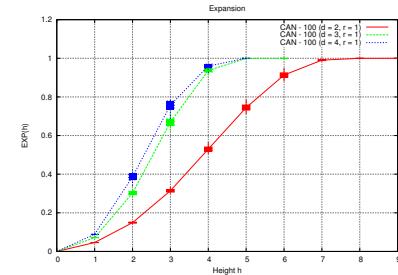
Configuring a new Network

```
CAN_NETWORK_NAME = CAN
```

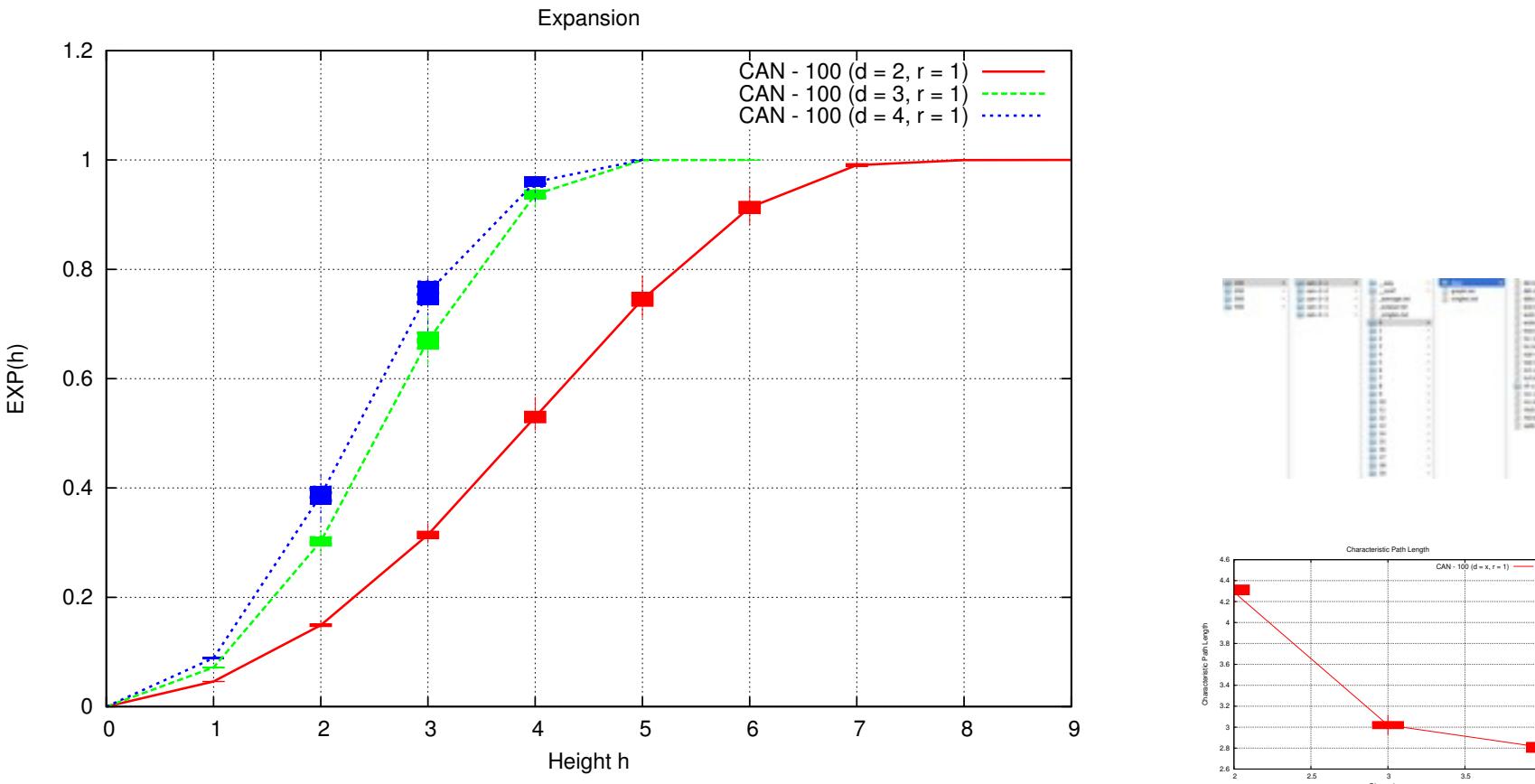
```
CAN_NETWORK_FOLDER = can
```

```
CAN_NETWORK_DIMENSIONS_NAME = Dimensions
CAN_NETWORK_DIMENSIONS_NAME_LONG = dimensions
CAN_NETWORK_DIMENSIONS_NAME_SHORT = d
```

```
CAN_NETWORK_REALITIES_NAME = Realities
CAN_NETWORK_REALITIES_NAME_LONG = realities
CAN_NETWORK_REALITIES_NAME_SHORT = r
```



Configuring a new Network



Configuring a new Network

```
CAN_NETWORK_NAME = CAN
```

```
CAN_NETWORK_FOLDER = can
```

```
CAN_NETWORK_DIMENSIONS_NAME = Dimensions
```

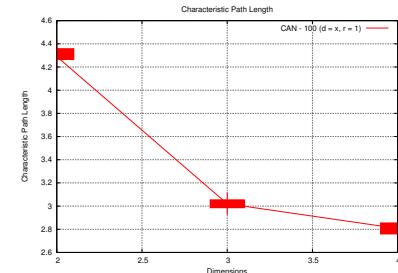
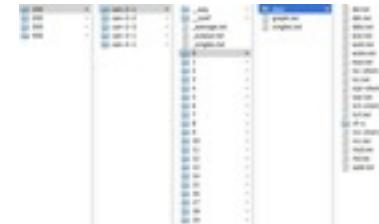
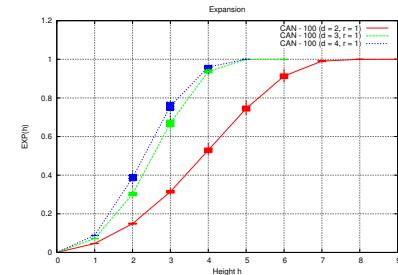
```
CAN_NETWORK_DIMENSIONS_NAME_LONG = dimensions
```

```
CAN_NETWORK_DIMENSIONS_NAME_SHORT = d
```

```
CAN_NETWORK_REALITIES_NAME = Realities
```

```
CAN_NETWORK_REALITIES_NAME_LONG = realities
```

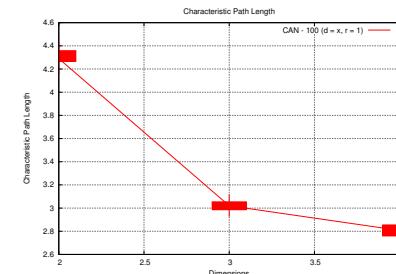
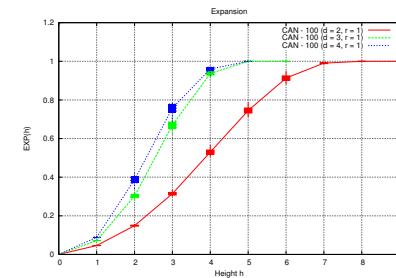
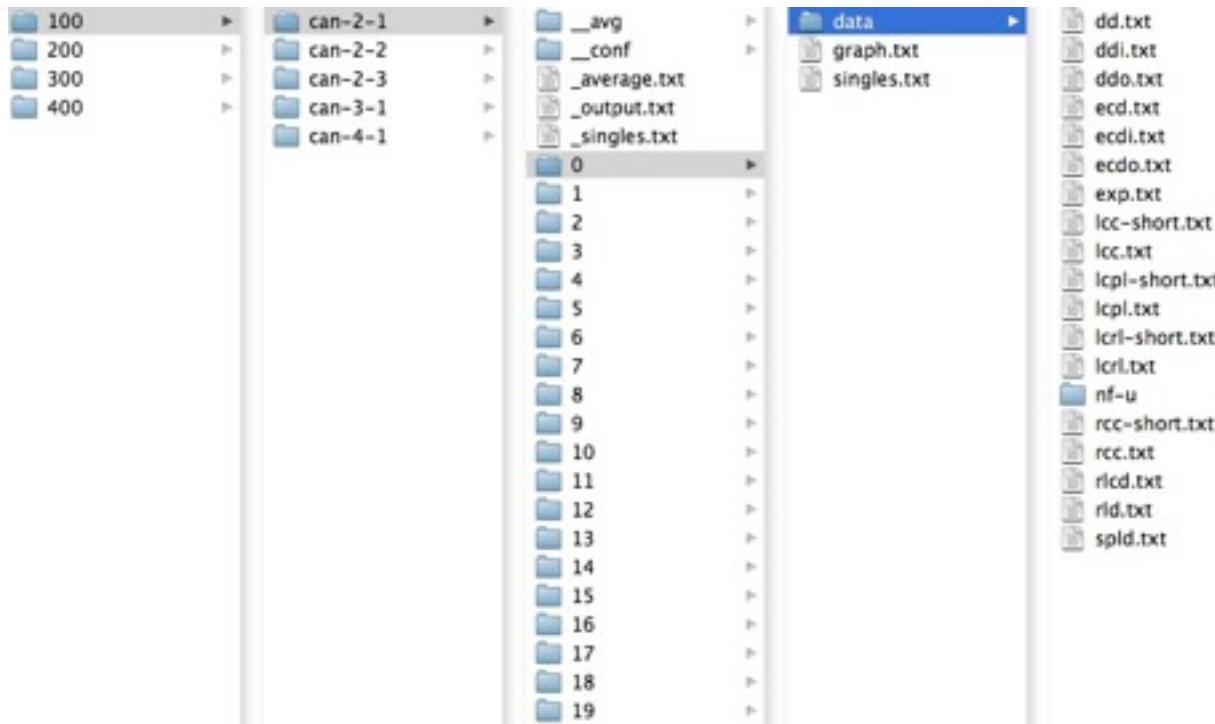
```
CAN_NETWORK_REALITIES_NAME_SHORT = r
```



Configuring a new Network



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Configuring a new Network

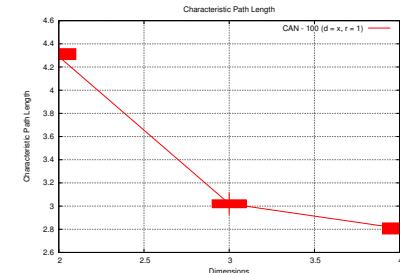
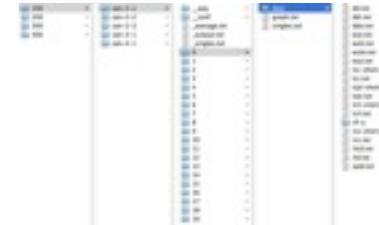
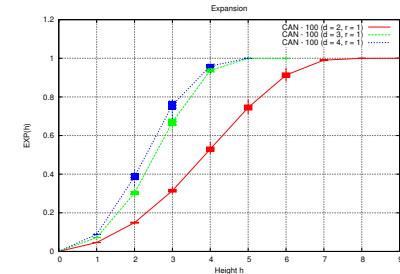


```
CAN_NETWORK_NAME = CAN
```

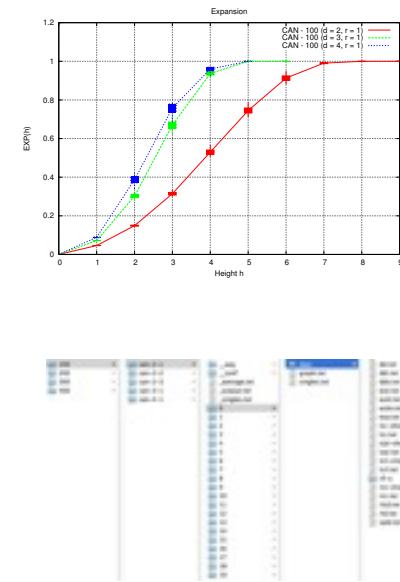
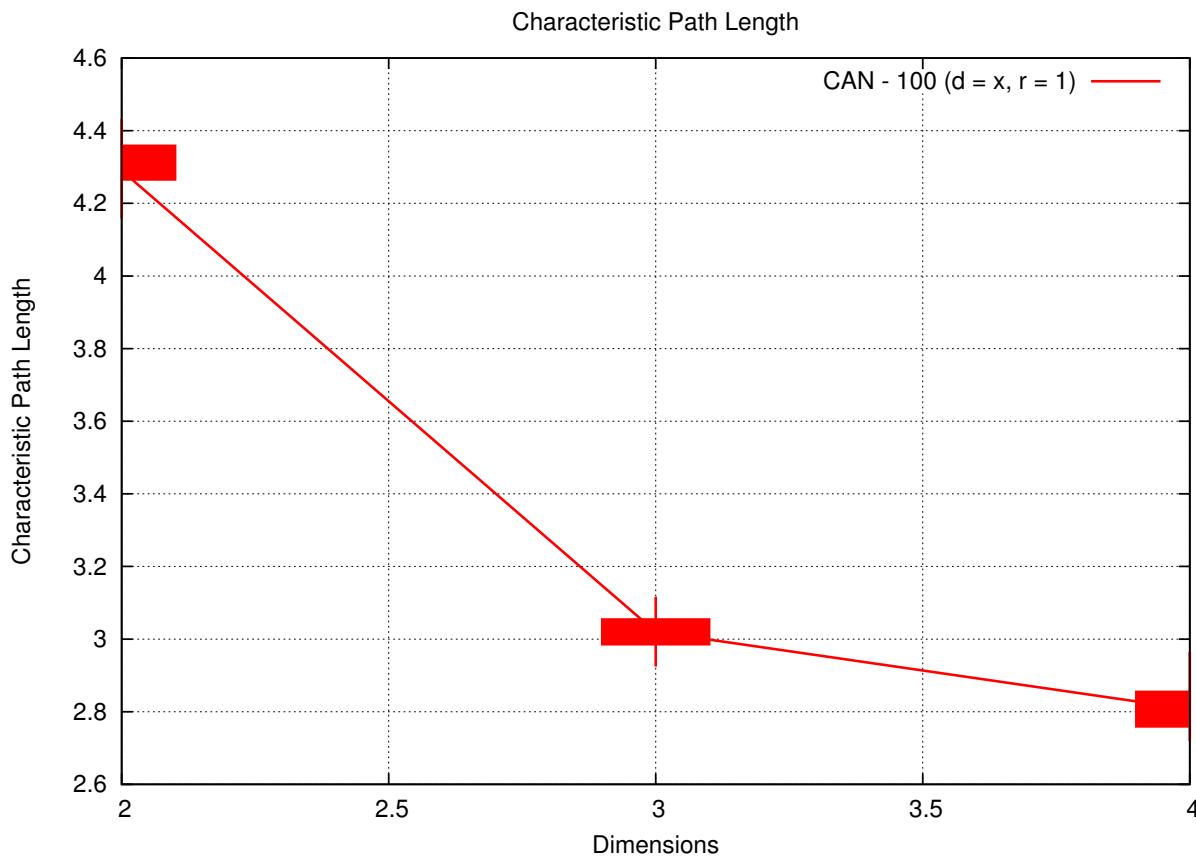
```
CAN_NETWORK_FOLDER = can
```

```
CAN_NETWORK_DIMENSIONS_NAME = Dimensions
CAN_NETWORK_DIMENSIONS_NAME_LONG = dimensions
CAN_NETWORK_DIMENSIONS_NAME_SHORT = d
```

```
CAN_NETWORK_REALITIES_NAME = Realities
CAN_NETWORK_REALITIES_NAME_LONG = realities
CAN_NETWORK_REALITIES_NAME_SHORT = r
```



Configuring a new Network



Implementing a new Metric

```
public class MyMetric extends MetricImpl implements Metric {  
    private int[] nod;  
    private int domn;  
    public MyMetric() { super("NOD"); }  
    public void computeData(Graph g) {  
        this.nod = new int[g.maxDegree + 1];  
        for (int i = 0; i < g.nodes.length; i++)  
            this.nod[g.nodes[i].degree]++;  
        this.domn = Util.maxIndex(this.nod);  
    }  
    public Value[] getValues(Value[] values) {  
        return new Value[] { new Value("NOD_DOMN", this.domn) };  
    }  
    public boolean writeData(String folder) {  
        DataWriter.write("NOD_NOD", folder, this.nod);  
        return true;  
    }  
}
```

Implementing a new Metric

```
public class MyMetric extends MetricImpl implements Metric {  
    private int[] nod;  
    private int domn;  
    public MyMetric() { super("NOD"); }  
    public void computeData(Graph g) {  
        this.nod = new int[g.maxDegree + 1];  
        for (int i = 0; i < g.nodes.length; i++)  
            this.nod[g.nodes[i].degree]++;  
        this.domn = Util.maxIndex(this.nod);  
    }  
    public Value[] getValues(Value[] values) {  
        return new Value[] { new Value("NOD_DOMN", this.domn) };  
    }  
    public boolean writeData(String folder) {  
        DataWriter.write("NOD_NOD", folder, this.nod);  
        return true;  
    }  
}
```

Implementing a new Metric

```
public class MyMetric extends MetricImpl implements Metric {  
    private int[] nod;  
    private int domn;  
  
    public MyMetric() { super("NOD"); }  
    public void computeData(Graph g) {  
        this.nod = new int[g.maxDegree + 1];  
        for (int i = 0; i < g.nodes.length; i++)  
            this.nod[g.nodes[i].degree]++;  
        this.domn = Util.maxIndex(this.nod);  
    }  
    public Value[] getValues(Value[] values) {  
        return new Value[] { new Value("NOD_DOMN", this.domn) };  
    }  
    public boolean writeData(String folder) {  
        DataWriter.write("NOD_NOD", folder, this.nod);  
        return true;  
    }  
}
```

Attributes

Implementing a new Metric

```
public class MyMetric extends MetricImpl implements Metric {  
    private int[] nod;  
    private int domn;  
    public MyMetric() { super("NOD"); }  
    public void computeData(Graph g) {  
        this.nod = new int[g.maxDegree + 1];  
        for (int i = 0; i < g.nodes.length; i++)  
            this.nod[g.nodes[i].degree]++;  
        this.domn = Util.maxIndex(this.nod);  
    }  
    public Value[] getValues(Value[] values) {  
        return new Value[] { new Value("NOD_DOMN", this.domn) };  
    }  
    public boolean writeData(String folder) {  
        DataWriter.write("NOD_NOD", folder, this.nod);  
        return true;  
    }  
}
```

Attributes

Constructor

Implementing a new Metric

```
public class MyMetric extends MetricImpl implements Metric {  
    private int[] nod;  
    private int domn;  
    public MyMetric() { super("NOD"); }  
    public void computeData(Graph g) {  
        this.nod = new int[g.maxDegree + 1];  
        for (int i = 0; i < g.nodes.length; i++)  
            this.nod[g.nodes[i].degree]++;  
        this.domn = Util.maxIndex(this.nod);  
    }  
    public Value[] getValues(Value[] values) {  
        return new Value[] { new Value("NOD_DOMN", this.domn) };  
    }  
    public boolean writeData(String folder) {  
        DataWriter.write("NOD_NOD", folder, this.nod);  
        return true;  
    }  
}
```

Attributes

Constructor

Computation

Implementing a new Metric

```
public class MyMetric extends MetricImpl implements Metric {  
    private int[] nod;  
    private int domn;  
    public MyMetric() { super("NOD"); }  
    public void computeData(Graph g) {  
        this.nod = new int[g.maxDegree + 1];  
        for (int i = 0; i < g.nodes.length; i++)  
            this.nod[g.nodes[i].degree]++;  
        this.domn = Util.maxIndex(this.nod);  
    }  
    public Value[] getValues(Value[] values) {  
        return new Value[] { new Value("NOD_DOMN", this.domn) };  
    }  
    public boolean writeData(String folder) {  
        DataWriter.write("NOD_NOD", folder, this.nod);  
        return true;  
    }  
}
```

Constructor

Attributes

Computation

Output

Configuring a new Metric



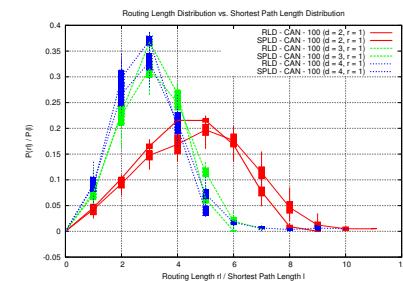
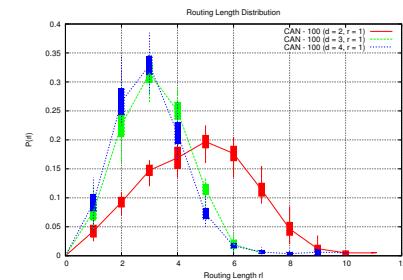
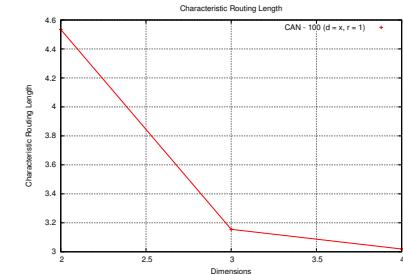
```
RL_CLASS = gtna.metrics.RoutingLength  
RL_NAME = Routing Length  
RL_SINGLES_KEYS = CRL  
RL_DATA_KEYS = RLD  
RL_SINGLES_PLOTS = CRL  
RL_DATA_PLOTS = RLD
```

```
CRL_SINGLE_NAME = Characteristic Routing Length
```

```
RLD_DATA_NAME = Routing Length Distribution  
RLD_DATA_FILENAME = RLD
```

```
CRL_PLOT_DATA = CRL  
CRL_PLOT_FILENAME = crl  
CRL_PLOT_TITLE = Characteristic Routing Length  
CRL_PLOT_Y = CRL
```

```
RLD_PLOT_DATA = RLD  
RLD_PLOT_FILENAME = rld  
RLD_PLOT_TITLE = Routing Length Distribution  
RLD_PLOT_X = Routing Length rl  
RLD_PLOT_Y = P(rl)
```



Configuring a new Metric



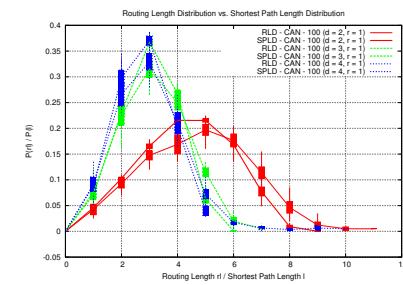
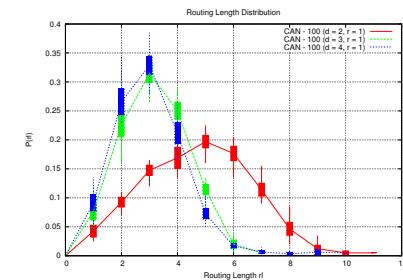
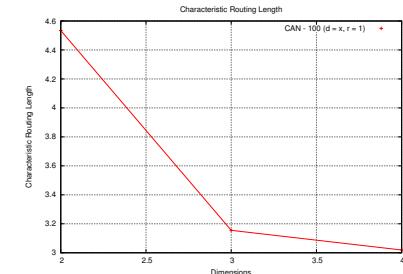
```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLE_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLE_PLOTS = CRL
RL_DATA_PLOTS = RLD
```

```
CRL_SINGLE_NAME = Characteristic Routing Length
```

```
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD
```

```
CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL
```

```
RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
```



Configuring a new Metric



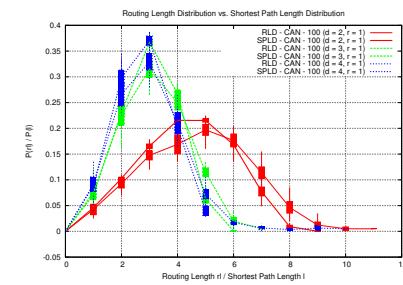
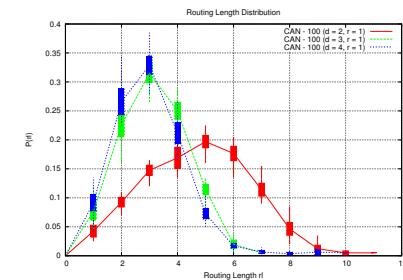
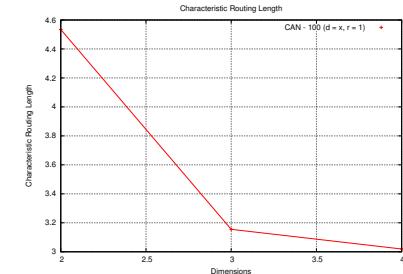
```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD
```

CRL_SINGLE_NAME = Characteristic Routing Length

```
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD
```

```
CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL
```

```
RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
```



Configuring a new Metric



```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD
```

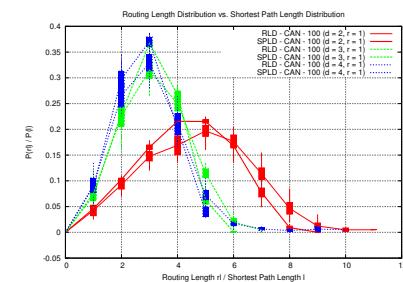
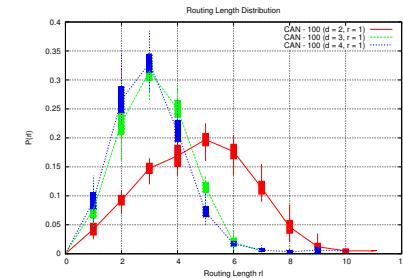
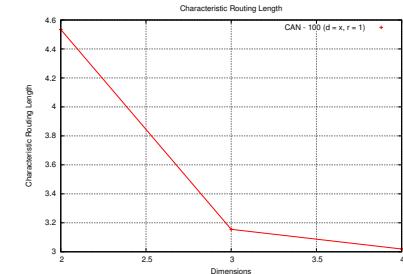
```
CRL_SINGLE_NAME = Characteristic Routing Length
```

```
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD
```

```
CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
```

```
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL
```

```
RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
```



Configuring a new Metric



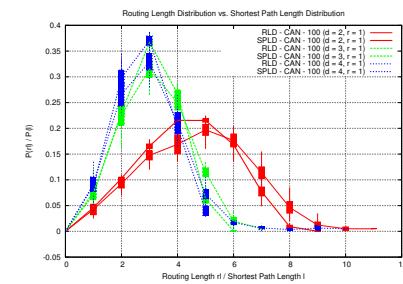
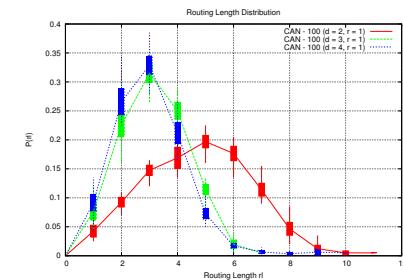
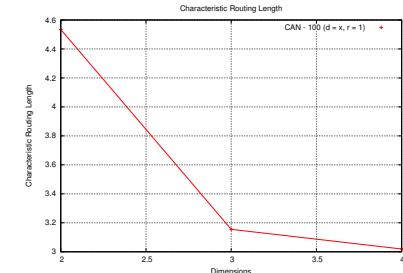
```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD
```

```
CRL_SINGLE_NAME = Characteristic Routing Length
```

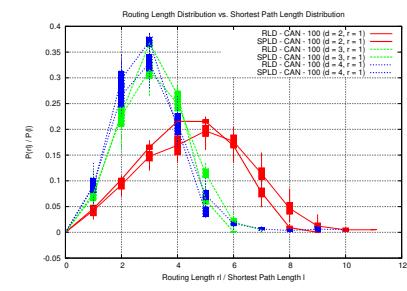
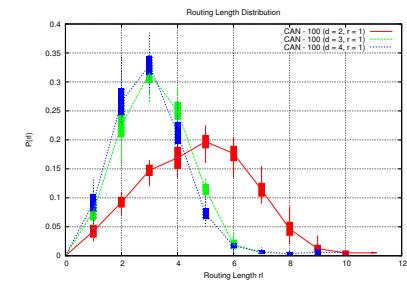
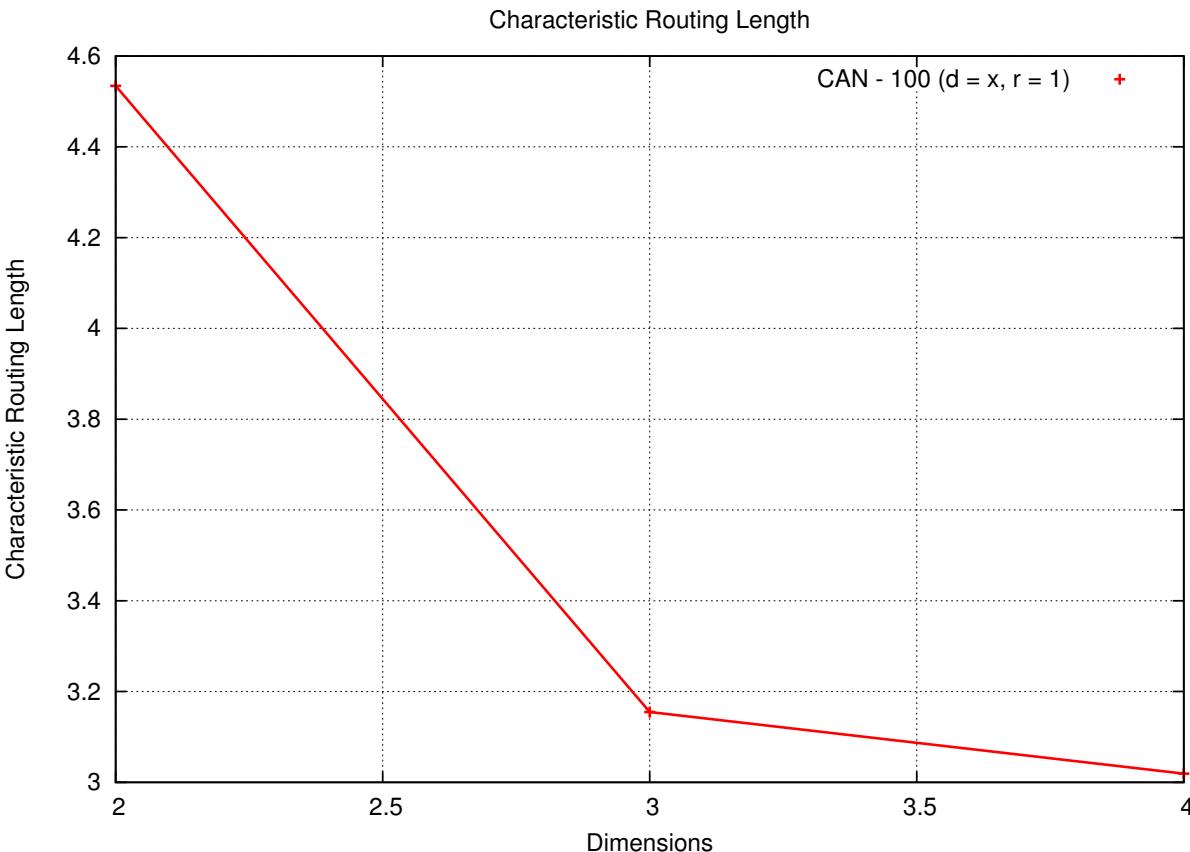
```
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD
```

```
CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL
```

```
RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
```



Configuring a new Metric



Configuring a new Metric



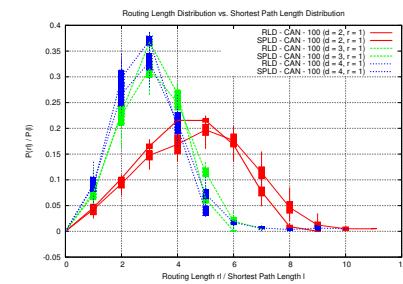
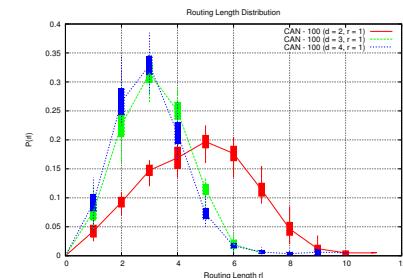
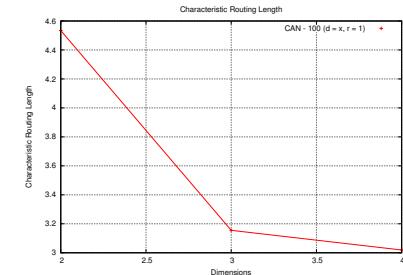
```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD
```

```
CRL_SINGLE_NAME = Characteristic Routing Length
```

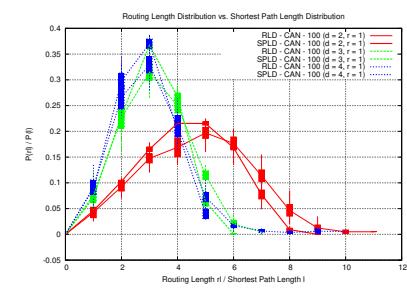
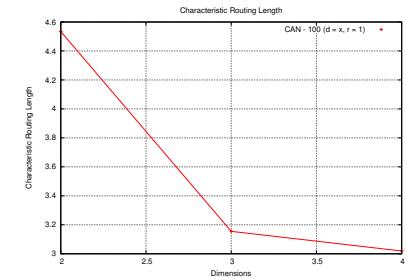
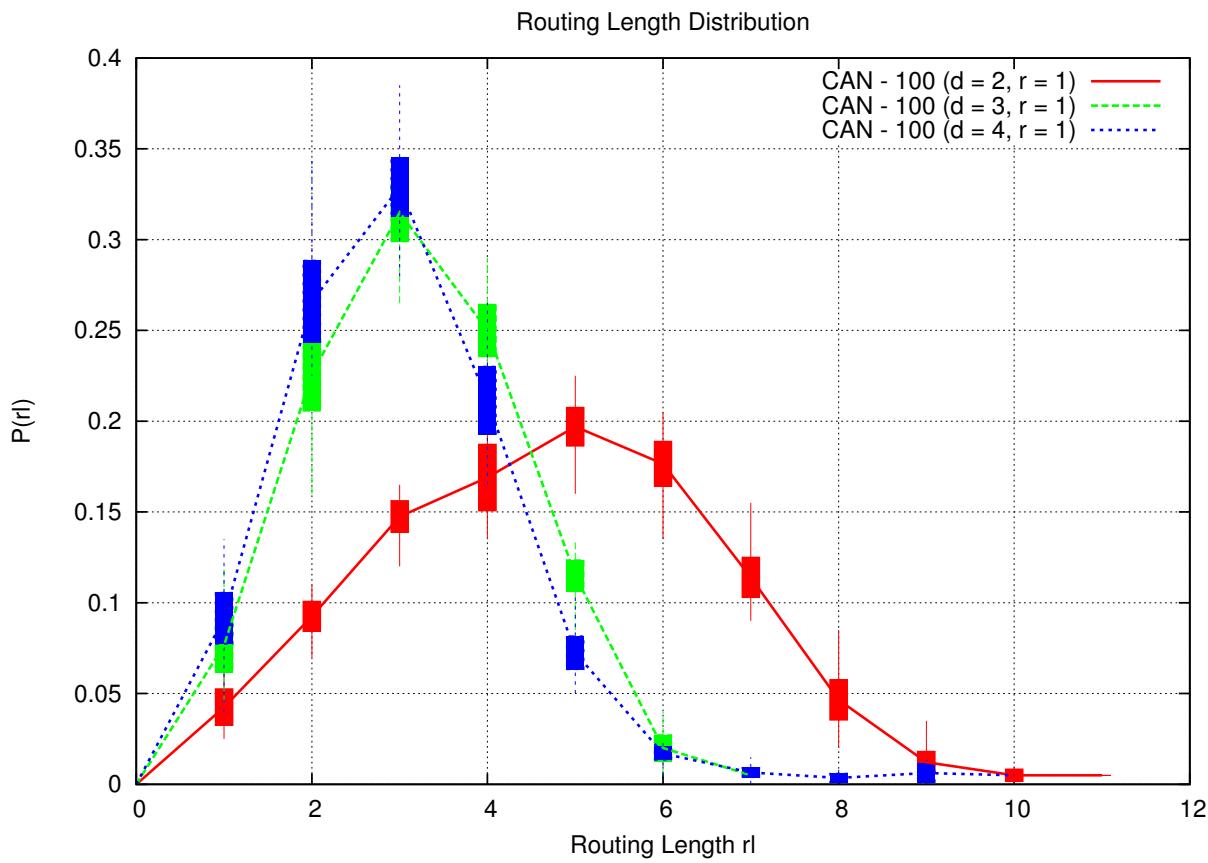
```
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD
```

```
CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL
```

```
RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
```



Configuring a new Metric



Configuring a new Metric



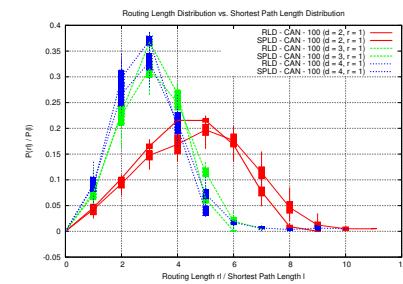
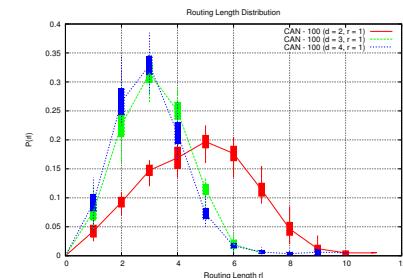
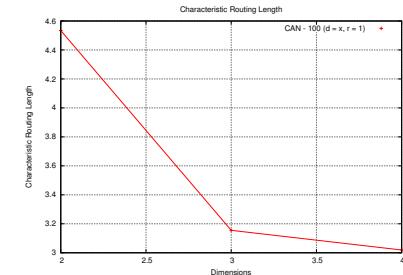
```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD
```

```
CRL_SINGLE_NAME = Characteristic Routing Length
```

```
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD
```

```
CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL
```

```
RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
```



Configuring a new Metric



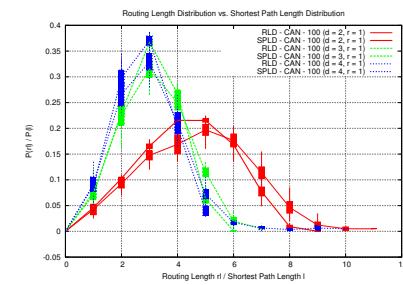
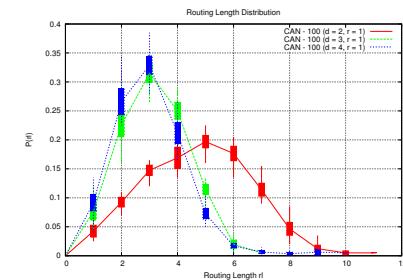
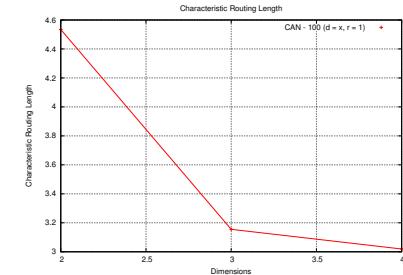
```
RL_CLASS = gtna.metrics.RoutingLength  
RL_NAME = Routing Length  
RL_SINGLES_KEYS = CRL  
RL_DATA_KEYS = RLD  
RL_SINGLES_PLOTS = CRL  
RL_DATA_PLOTS = RLD
```

```
CRL_SINGLE_NAME = Characteristic Routing Length
```

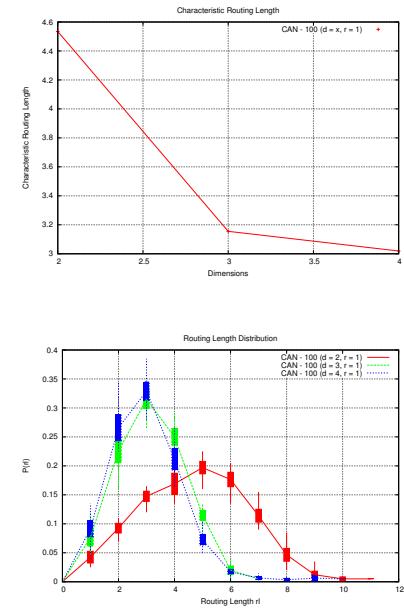
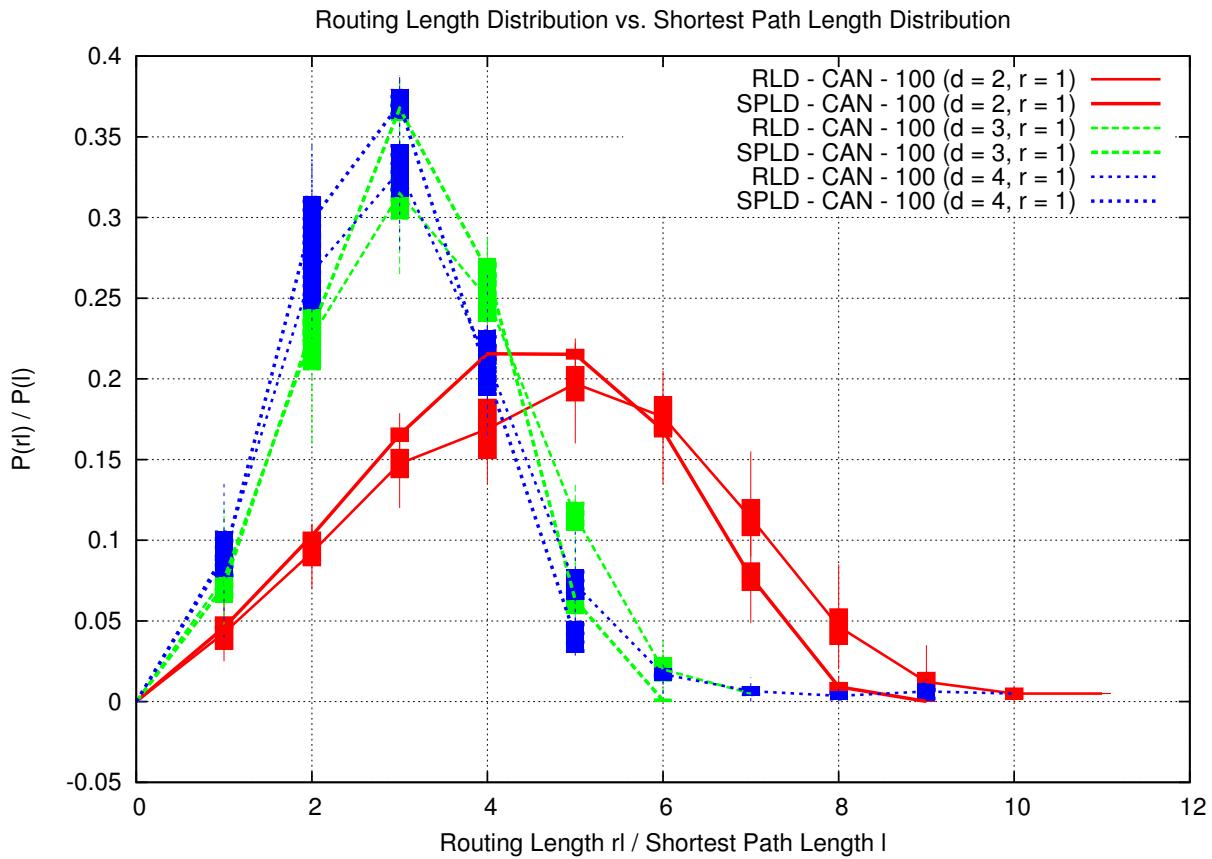
```
RLD_DATA_NAME = Routing Length Distribution  
RLD_DATA_FILENAME = RLD
```

```
CRL_PLOT_DATA = CRL  
CRL_PLOT_FILENAME = crl  
CRL_PLOT_TITLE = Characteristic Routing Length  
CRL_PLOT_Y = CRL
```

```
RLD_PLOT_DATA = RLD, SPLD  
RLD_PLOT_FILENAME = rld-vs-spld  
RLD_PLOT_TITLE = RLD vs. SPLD  
RLD_PLOT_X = Routing Length rl / Shortest Path Length l  
RLD_PLOT_Y = P(rl) / R(l)
```



Configuring a new Metric



Configuring a new Metric



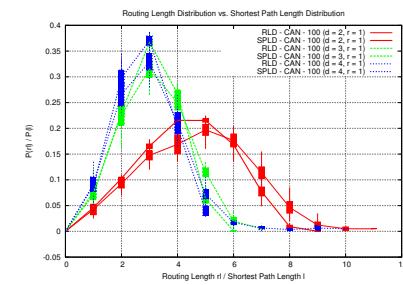
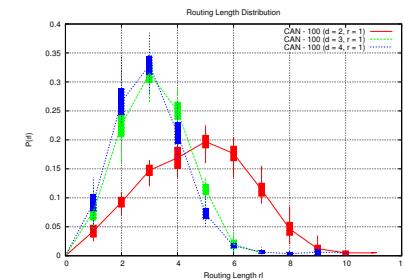
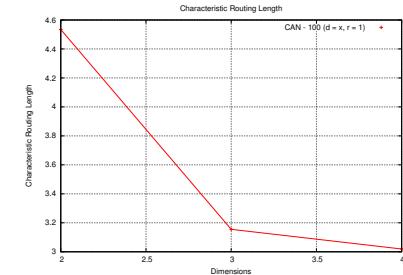
```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD
```

```
CRL_SINGLE_NAME = Characteristic Routing Length
```

```
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD
```

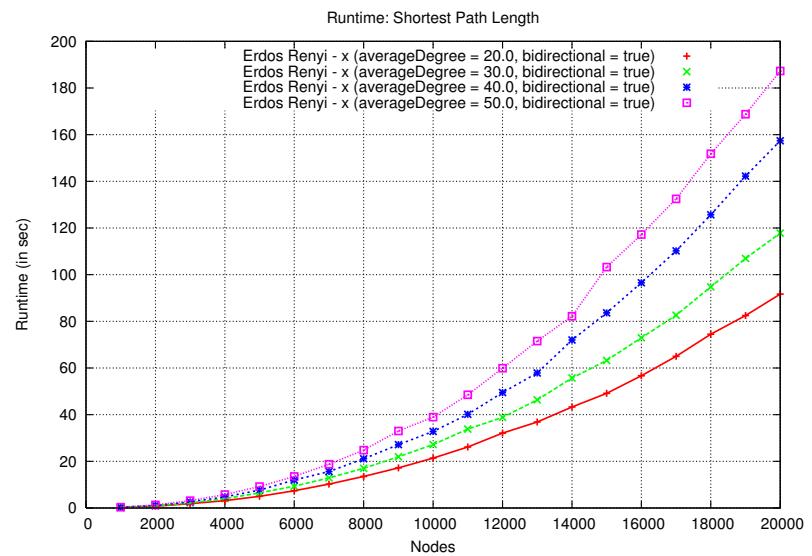
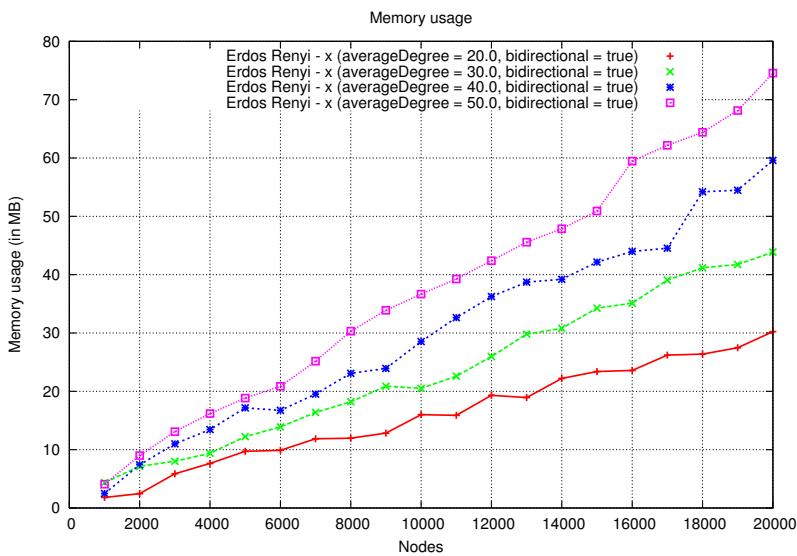
```
CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL
```

```
RLD_PLOT_DATA = RLD, SPLD
RLD_PLOT_FILENAME = rld-vs-spld
RLD_PLOT_TITLE = RLD vs. SPLD
RLD_PLOT_X = Routing Length rl / Shortest Path Length l
RLD_PLOT_Y = P(rl) / R(l)
```



6. Evaluation

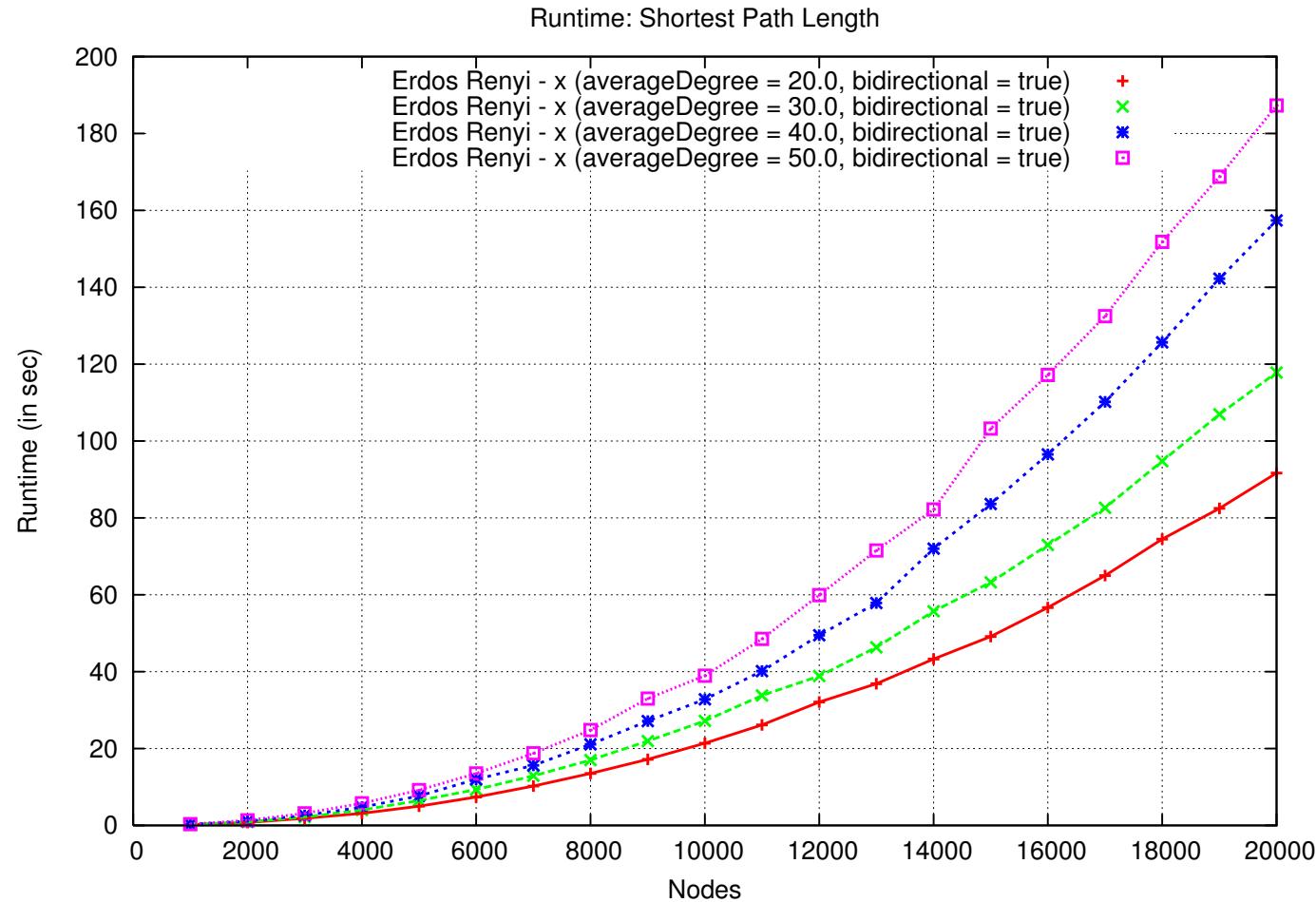
Runtime and Memory Usage Evaluation



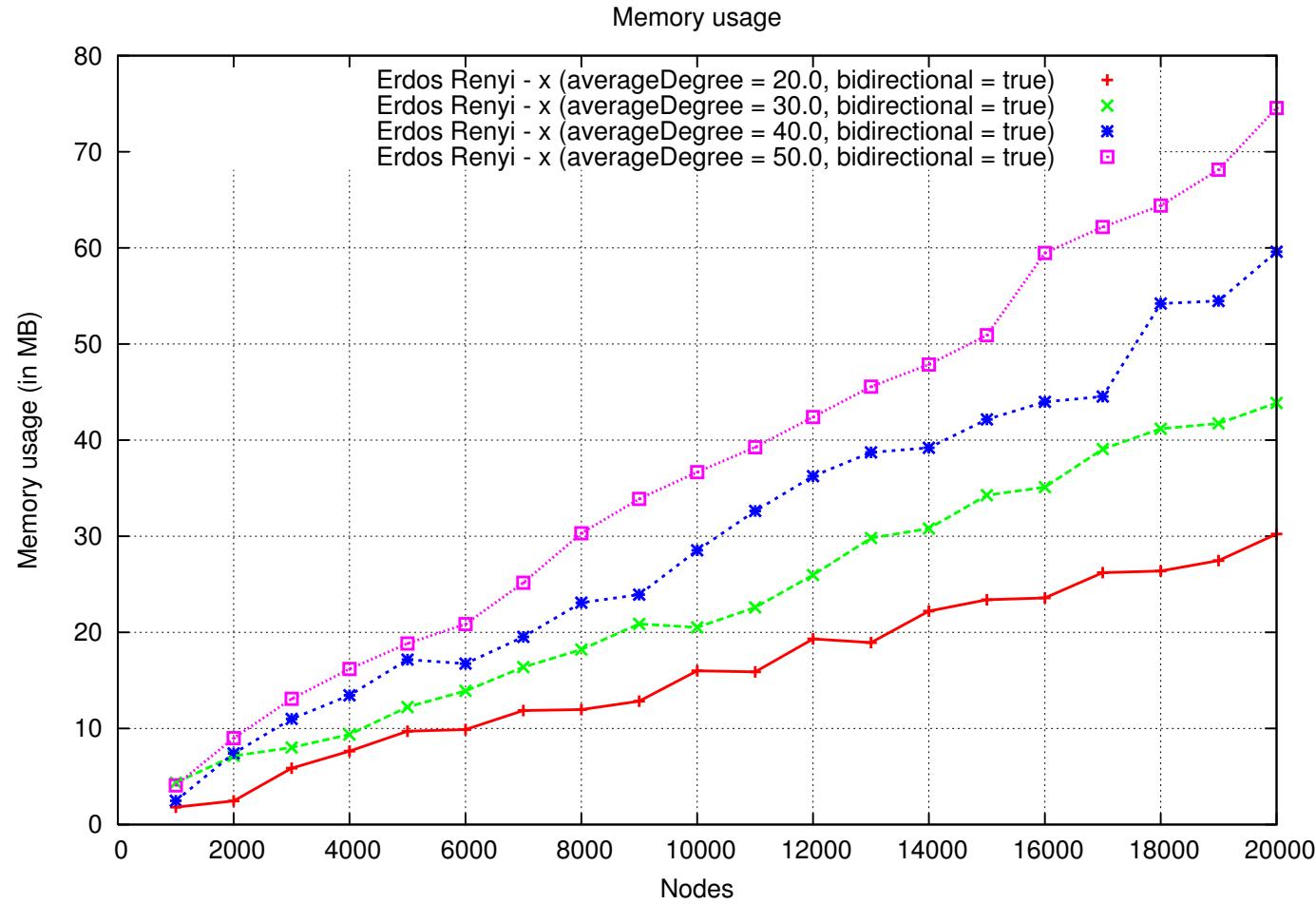
Setup

- Computer
 - 2.4 GHs Intel Core 2 Duo, 2 GB of Memory
 - MAC OS 10.5.8, Java version 1.5.0_22
- Networks
 - Erdös Rényi random graph
 - $d_{avg} \in \{ 20, 30, 40, 50 \}$
- Runtime: Shortest Path Length
- Memory usage: all metrics

Runtime



Memory Usage



Requirements

1. Extensability
2. Run on regular desktop computers
3. Evaluate networks with > 20.000 nodes
4. Require less than 2 GB of memory

Requirements

1. Extensability ✓
2. Run on regular desktop computers
3. Evaluate networks with > 20.000 nodes
4. Require less than 2 GB of memory

Requirements

1. Extensability 
2. Run on regular desktop computers 
3. Evaluate networks with > 20.000 nodes
4. Require less than 2 GB of memory

Requirements

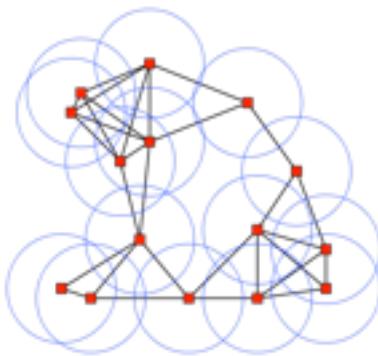
1. Extensability 
2. Run on regular desktop computers 
3. Evaluate networks with > 20.000 nodes 
4. Require less than 2 GB of memory

Requirements

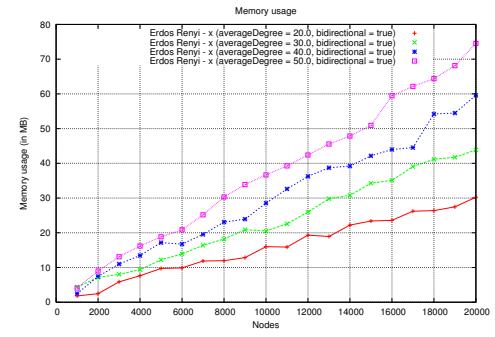
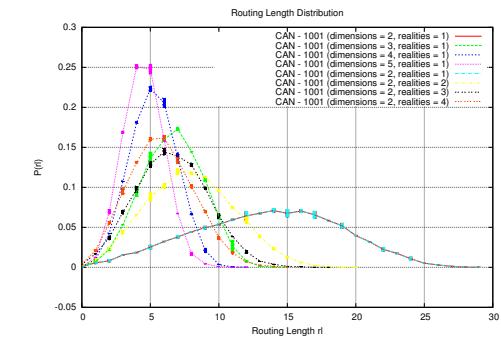
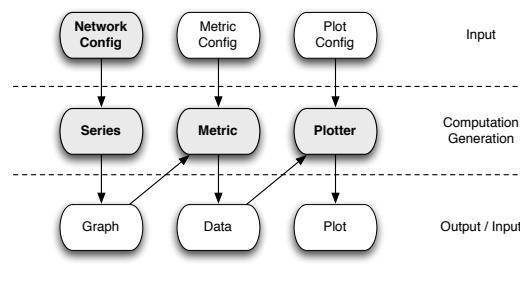
1. Extensability 
2. Run on regular desktop computers 
3. Evaluate networks with > 20.000 nodes 
4. Require less than 2 GB of memory 

7. Summary and Outlook

Summary and Planned Extensions to GTNA



```
public class MyNetwork extends NetworkImpl implements Network {
    private boolean p1;
    public MyNetwork(int nodes, boolean p1) {
        super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});
        this.p1 = p1;
    }
    public Graph generate(){
        Timer timer = new Timer();
        Node[] nodes = Node.init(this.nodes());
        Edges edges = new Edges(nodes, 100);
        for(int i=1; i<nodes.length; i++){
            edges.add(nodes[0], nodes[i]);
            if(this.p1)
                edges.add(nodes[i], nodes[0]);
        }
        edges.fill();
        return new Graph(this.description(), nodes, timer);
    }
}
```



Summary

- Graph-theoretic analysis helps understand complex networks
- Often difficult task
 - adapting specific algorithms to special network formats
- GTNA - Graph-Theoretic Network Analyzer
 - Java-based framework
 - Easily extendable
 - Many network generators and metric implementations exist

Outlook



- Implementing additional metrics
- Providing Internet topology c
- Analysis of network dynamics
- Graphical user interface

Motifs
Role-to-role connectivity profiles

PFP - Positive-Feedback Preference model
PARG - Parallel Addition and Requiring Growth model

Different procedures for topology generation
(join, crash, leave)

Configuration
Plot-Organization

GTNA

A Framework for the Graph-Theoretic Network Analysis



Benjamin Schiller Dirk Bradler Immanuel Schweizer
Max Mühlhäuser Thorsten Strufe



GTNA

Graph-Theoretic Network Analyzer